ON FAST ALGORITHM AND VLSI DESIGN OF FINITE COMPUTATIONAL STRUCTURES

By

ROM-SHEN KAO

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1991

Dedicated to

my parents

ACKNOWLEDGMENTS

In the years leading to this dissertation, I have had the great pleasure of working for and with an adviser who provided an environment conducive to learning. I would like to take this opportunity to thank my advisor, Dr. Fred J. Taylor.

I would also like to thank Dr. Y. C. Chow, Dr. H. Lam, Dr. M. Law, and Dr. J. Principe for serving on my supervisory committee.

Special thanks to my fellow students, who have offered useful advice. Thanks go to Ahmad Ansari, Glenn Dean, Jun Li, Jon Mellott, Jeremy Smith, Eric Strom, Glenn Zelniker, and especially, John Ruckstuhl, for his courtesy and professional work on setting up the SUN systems.

I would also like to thank those I have come to know at the University of Florida. They make my years at the university a unique and memorable experience.

This work would not be possible without the encouragement and support from my wife, Jeannie.

Finally, I would like to thank my parents for teaching me the philosophy of life and the value of education, and most of all for their love, patience, and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

KEY TO SYMBOLS AND ABBREVIATIONS

SYMBOL	EXPLANATION
DFT	Discrete Fourier Transform
FFT	fast Fourier transform
IFFT	inverse fast Fourier transform
$\mathbf{N}$	the set of natural numbers (= positive integers)
$\mathbf{Z}$	the set of integers
$\mathbf{Q}$	the set of rational numbers
$\mathbf{R}$	the set of real numbers
$\mathbf{C}$	the set of complex numbers
$S_1 \times \ldots \times S_n$	the set of all n-tuples $(s_1, \ldots, s_n)$ where $s_i \in S_i$ for $1 \leq i \leq n$
S^n	the set of all n-tuples $(s_1, \ldots, s_n)$ where $s_i \in S_i$ for $1 \leq i \leq n$
$\lvert S \rvert$	the order or cardinality (=number of elements) of the finite set S
$\lfloor t \rfloor$	the greatest integer $\leq t \in \mathbf{R}$
$\gcd(k_1, \ldots, k_n)$	the greatest common divisor of $k_1, \ldots, k_n$
$\mathrm{lcm}(k_1, \ldots, k_n)$	the least common multiple of $k_1, \ldots, k_n$
$\binom{k}{i}$	binomial coefficient
$\phi(n)$	Euler's function of n
A^T	the transpose of the matrix A
$\det(A)$	the determinant of the matrix A
$<a>$	the cyclic group generated by a
(a)	the principal ideal generated by a

R/J	the residue class ring of the ring modulo the ideal J
$\mathbf{Z}_n$	the group of integers modulo n
$\mathrm{Z}/(n)$	the ring of integers modulo n
$R[x]$	the polynomial ring over the ring R
$\deg(f)$	the degree of the polynomial f
$Q_n(x)$	the nth cyclotomic polynomial
$K(M)$	the extension of K obtained by adjoining M
$[L{:}K]$	the degree of the field L over K
F_p, GF(p)	the finite field of order p
F_p^*	the multiplicative group of nonzero elements of F_p
$\mathrm{Tr}(\alpha)$	the trace of the element α of the field F
$\mathrm{N}(\alpha)$	the norm of the element α of the field F
$\log_b(a)$	the discrete logarithm of a with respect to the base b
$\exp_b(r)$	the discrete exponential function of r with respect to the base b
♦	end of proof, end of example, end of remark

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

ON FAST ALGORITHM AND VLSI DESIGN OF FINITE COMPUTATIONAL STRUCTURES

By

Rom-Shen Kao

May 1991

Chairman: Dr. Fred J. Taylor
Major Department: Electrical Engineering

This dissertation examines some properties of the finite computational structures such as finite groups, rings, and fields, and extends its applications to develop fast algorithms and to build high-speed low-complexity very large scale integrated circuit (VLSI) systems for digital signal processing.

To achieve high speed signal processing, the finite computational structures are receiving growing attention because of its ability to support high-speed integer arithmetic. In this dissertation, various fundamental algebraic theories are applied to the development of a finite computational system. A finite polynomial ring structure is addressed to expedite and simplify the multiplication of polynomials. The applications of this polynomial structure are in the areas of short-block length cyclic convolution and complex number arithmetic which originates from the concept of algebraic integer approximation. A new algorithm of finite field transforms with cyclic convolution property (CCP) is investigated. This algorithm combines abstract algebra concepts which includes normal basis representation and conjugacy property with factor-type fast Fourier transform algorithms (FFT) to expedite finite field transforms. Finally, a detailed design of a transform butterfly module and a third-order polynomial RNS (PRNS) processor are presented. It is shown that compact, economical, and high-performance design is feasible when based on a VLSI architecture.

CHAPTER 1
INTRODUCTION

1.1 Overview

The theory of the finite computational structures, finite groups, rings, or fields, is a branch of modern algebra which has climbed to the forefront of investigational activity during the past 50 years. Reflecting the work of such eminent mathematicians as Euclid, Pierre de Fermat, Leonard Euler, Joseph-Louis Lagrange, Carl Friedrich Gauss and Evariste Galois, this field of study contains diverse applications in combinatorics, coding theory, and mathematical studies of switching circuits. A flurry of recent activity which ensued in the development of interesting, computationally efficient algorithms coupled with the advent of high-speed high-density digital computing devices make the field well suited to applications in data communications, error control codes, speech processing, radar/sonar signal processing, and image processing.

Arithmetic operations in finite computational structures play an important role in convolving and transforming digital signal, as well as in encoding/decoding error control codes, where sampled data indices or amplitudes are treated in a number-theoretic manner. Modular arithmetic utilizes mathematical operations involving multiplication, power-raising, multiplicative inversion, convolution, and transform. In terms of application, these operations are among the most complex and time-consuming. Hence, implementation of a fast algorithm which serves to minimize either the cost or the computation time of the circuits required to perform these operations results in a considerable improvement within the field of applications.

While some computational algorithms are more popular than others, users generally attribute a higher value to the most efficient version of these algorithms. One concept central to many of these algorithms is the Chinese remainder theorem (CRT) in residue number system (RNS), a method which dates back to antiquity and involves the partitioning of a large problem into a number of smaller subproblems. By strengthening and reestablishing the threads spun from classical mathematics, CRT opened numerous applications for applied mathematicians and arithmetic complexity theorists, and research took a necessary step toward digital signal processing (DSP). Another DSP-related algorithm is number theoretic transform (NTT) which is defined over finite fields and rings of integers with all arithmetic performed modulo an integer. These transforms which have cyclic convolution property (CCP) are shown to be advantageous to digital convolution based applications such as digital filtering, correlation studies, and the multiplication of very large integers.

The primary components of a DSP system include a multiplier or multiplier/accumulator, program and data memory, and a controller with some form of register files. Since high levels of integration are the current trend in system design, the existing general purpose processors may not meet new demands. In recent years, the progress of Very Large Scale Integrated circuit (VLSI) technology has been widely applied in signal processing. In the custom design arena, higher density and more sophisticated gate-array designs realize the dream of placing entire systems on a single chip which can contain on the order of 100,000 logical gates. Theory of algorithms produce a means of efficiently organizing these gates. Thus, the integration of fast arithmetic and mathematical algorithms for signal processing is a key factor in effecting a break through of the restricted development in electronic devices technology. This dissertation introduces novel concepts in number system application and discusses the development of their associated systems.

1.2 Problem Statement and Objectives

Applications in digital filtering, correlation studies, radar matched filtering, and the multiplication of very large integers are based on digital convolution, which can be implemented most efficiently by NTT with some constraints. The arithmetic required to accomplish the NTT is exact and involves additions, subtractions, and bit shifts. As in the case of DFT, fast algorithms exist for the NTT. The family of NTT includes Fermat, Mersenne, Rader, pseudo-Fermat, pseudo-Mersenne, complex Mersenne, and complex Fermat transforms [McC79, Ell82]. They are truly digital transforms and their implementation involves no round-off error. The implementation of NTT required multiplications (scaling) and additions/subtractions is such a time-consuming (or hardware intensive) task that any advantage inherited from the cyclic convolution property is eventually offset.

Recent attention has been focused on situations requiring data manipulation over complex fields. The problem of approximating complex numbers by elements of algebraic integers also has been investigated by many researchers[Gam85, Coz85]. The operations in the ring of algebraic integers are translated to the operations of polynomials by the polynomial residue number systems (PRNS). The modular number system admits an unusual representation of complex data, which leads to a complete decoupling of the real and imaginary channels, thereby simplifying complex multiplication and providing error isolation between the channels. Unfortunately, isomorphic mappings between complex number and PRNS domains suffer from a nontrivial transform problem which eventually precludes the inherent advantages of the PRNS approach.

The maturing of silicon technology and the sophisticated CAD/CAE tools have allowed for the economical development of semicustom and custom application-specific integrated circuit (ASIC) chips. The concurrence, modularity, and regular interconnection of the finite computational structures, such as the RNS system, are the main properties of an efficient VLSI system. Taking all these merits, Taylor and Kao[Kao87] have developed a powerful complex ALU, which is based on a quadratic RNS (QRNS), and proven the concept using 4187 gates of the GE gate-array design. Based on these observations, the coupling of the finite computational structures with the recent advances in VLSI technology suggests an efficient implementation of many signal processing algorithms.

While DSP researchers have responded to requirements for high speed and low cost, some levels of parallelism and low complexity modules are of primary concern. The trend of VLSI system design utilizes high levels of integration based on efficient algorithms. By investigating signal processing algorithms and mathematical theory, their close relationship looks so promising that a fundamental research effort could no longer be ignored. However, a successful investigation of this relationship cannot be accomplished without addressing the following problems:

1) The construction of fast algorithms for the applications of real-time signal processing from basic mathematical theories such as number theory and abstract algebra;

2) The development of an efficient transform method of mapping complex numbers to a new representation which provides parallelism and modular arithmetic capability:

Conventional complex signal processing requires the formulation of real and imaginary channels and the special handling of cross-product terms for complex multiplication. The efficient implementation of complex digital arithmetic has become increasingly important because many signal processing applications require processing of complex signals with complex digital filters;

3) The development of techniques to reduce the computational complexity of finite field transforms:

With the advantages of truly digital and roundoff-free transforms, the implementation of a cyclic convolution system becomes a promising task. However, the finite field transform is still an awkward mapping without special treatments; and

4) The development of a highly integrated arithmetic system which is equipped with fast, compact decomposed channels:

Speed and density always are conflicting factors in integrated-circuits (IC) technology. It seems advantageous to construct arithmetic systems, which provide the highest performance in terms of cost, speed and packaging, by implementing high-speed lower density devices.

The resolution of these problems involves the primary objectives of this research, which are as follows:

1) Achieve effective and fast algorithms of modular arithmetic and finite field operations to accomplish the development of polynomial RNS and finite field transform;

2) Apply the theories and algorithms to the development of a high-throughput low-complexity finite field transform system for digital convolution applications, and a simple isomorphic mapping pipeline polynomial RNS machine for complex arithmetic applications; and

3) Develop a systematic methodology for the efficient VLSI implementation of the developed architectures.

1.3 Dissertation Organization

This dissertation presents both the fast algorithm and the VLSI design as a solution to the problems of arithmetic computations, circular convolutions, and the discrete Fourier transform in finite computational structures with an emphases on finite fields or the so-called Galois fields. The work is principally based on modern algebra; while some of the well known theorems are shown without proof, most of the definitions are illustrated in the appendix for reference.

The mathematical framework from which this research effort stems is presented in Chapter 2. This section focuses on the mathematical preliminaries which include modular arithmetic, the Chinese remainder theorem, residue polynomial theory, and the finite computational structures of groups, rings and fields. The materials covered in Chapter 2 are based on text book discussions provided by McClellan and Rader[McC79], Lidl and Niederreiter[Lid86], and MacWilliams and Sloane[Mac77]. In order to reinforce the concepts of number theory principles, concrete examples are inserted at appropriate points in the text.

Computations in finite fields are developed in Chapter 3 which begins by presenting table-lookup methods for small finite field arithmetic involving discrete logarithm and exponentiation problems, while the nontrivial addition operation in logarithm form of finite fields is solved by applying Zech's logarithm method. Since the memory lookup method appears unrealistic for large fields, the discussion turns to an investigation of the sequential and parallel design approaches. These designs are based on various basis representations of finite field elements which include primal, dual, and normal basis representation. The normal basis structure is suggested as the most efficient model for calculation of the fast finite field transform. The chapter concludes with an investigation of the cyclic convolution property within a finite field.

Chapter 4 demonstrates various transforms within finite computational structures using number theoretic concepts for error-free and fast computation. A brief description of Fermat number transform (FNT) and Mersenne number transform (MNT) over integer rings are presented. These transforms are also applicable over extension fields which include the second-order extension field, GF(p^2), and the higher-order extension field, GF(p^m). A significant feature of the RNS is the ability to perform complex arithmetic efficiently. Several variations of the RNS for complex integers such as the most notably quadratic RNS and its extended version, polynomial RNS are investigated. Finally, finite field transforms and their algebraic properties are developed. This entire section sets the ground work for the system developments found in later chapters.

Drawing upon the results of modular arithmetic, complex arithmetic based on extension fields, and the finite field transforms all presented in previous chapters, systems of the polynomial RNS and fast finite field transform are developed in the following chapters. Parallel to the conventional discrete Fourier transform (DFT) in complex number fields, Chapter 5 introduces another system which details the involvement of a fast algorithm in computing DFTs within a finite field. This is accomplished by using the cyclotomic coset properties of finite fields to the intermediate stages of the transform iterations. Further computational savings in the transform are obtained by applying the cyclic shift to the evaluated element in a cyclotomic coset, where elements are represented in normal basis. During the sum-of-product operations within the evaluation of a coset leader of the intermediate stages, the field elements cannot maintain within the normal basis representation. Thus, a novel basis-change algorithm is applied to expedite the evaluation of the sum-of-products and maintain the intermediate result in the normal basis representation.

An in-depth discussion of previous theories regarding the implementation of the polynomial RNS arithmetic in the VLSI system along with their subsequent performance analyses is provided in Chapter 6. Design models considered for the pipeline third-order polynomial RNS processor based on Fermat number transform include the multiplier-free isomorphic mappings with fast Fourier transform expedition and the high speed modular multiplier. In terms of logic simulation and timing analysis, verification of this work is provided by the HP Design Capture System. Furthermore, a silicon layout of the 5-bit cubic polynomial RNS processor is implemented on the MAGIC computer-aided design (CAD) tool which is commensurate with the 2-micron double-metal complementary metal oxide semiconductor (CMOS) technology.

Chapter 7 presents the conclusions readily established by this research effort and offers a sense of direction for future research endeavors.

CHAPTER 2
FINITE COMPUTATIONAL STRUCTURES

2.1 Introduction

This chapter investigates the fundamental properties of finite computational structures pertaining to groups, rings and fields. The discussion begins by exploring the concepts of congruence and residue reduction. The foundations of number theory stemming from Euler's theorem and the concept of primitive roots are also introduced. Discussion of the Chinese remainder theorem in relation to both integers and polynomials leads to the realization that polynomial algebra has a close relationship to the digital convolution. The mathematical application of this relationship to a polynomial over fields induces the construction of finite extension fields. The properties of finite field such as power forming to a characteristic, minimal polynomials, conjugates, and the basis representation of a field element in an extension field over its ground field are also introduced.

2.2 Algebraic Foundations

The trend of VLSI system design utilizes high levels of integration based on efficient algorithms. Logically sound algorithms can be viewed as elegant algebraic identities. To construct these algorithms, one must be familiar with the powerful structures of number theory and of modern algebra. The structures containing the set of integers, polynomial rings, and finite fields play an important role in the design of signal processing algorithms. For instance, although it is most familiar within a complex field, a discrete Fourier transform can be defined in any field. In order to gain a better insight of the mathematical theories which lead to this application, a brief review of finite mathematical structures is provided along with a discussion of integer rings, residue polynomials, and finite fields.

Groups. A group is defined as a mathematical abstraction of an algebraic structure which appears frequently in various concrete forms. In general, let S be a set and let $S \times S$ denote the set of all ordered pairs (s, t) with $s, t \in S$. Then, a mapping from $S \times S$ into S is called a binary operation on S. The closure property of an operation requires that the image of (s, t) must be in S. A group G is a set which possesses a binary operation satisfying the properties of associativity, identity, and inverses (APPENDIX A D1.1). Furthermore, Groups with commutative property are called commutative group or abelian groups. A group that has a finite number of elements is defined as a finite group. The number of elements in a finite group G is called the order of G, denoted by $|G|$. An example of finite abelian groups are shown in Figure 2.1. Although these groups are represented by two different notations, it is

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\oplus$	e	g_1	g_2	g_3	g_4
e	e	g_1	g_2	g_3	g_4
g_1	g_1	g_2	g_3	g_4	e
g_2	g_2	g_3	g_4	e	g_1
g_3	g_3	g_4	e	g_1	g_2
g_4	g_4	e	g_1	g_2	g_3

Figure 2.1 Example of Abelian Finite Groups

essentially the same group shown twice. In general, any two algebraic systems having the same structure but different notations are called isomorphic.

A group is defined as cyclic when all elements of the group can be generated from one element of the group in the form a, $a*a$, $a*a*a$, . . . , where $*$ denotes the binary operation in the group. All cyclic groups of the same order are isomorphic to one another.

Rings. A ring R is an abstract set that is an abelian group having the property of performing another operation. Formally, a ring (R, +, ·) is a set R which contains two binary operations, such that: R is an abelian group with respect to '+', the operation '·' is associative, and the distributive laws hold (APPENDIX A D2.1). A commutative ring is one in which the operation '·' is commutative, also a ring with identity is one that has an identity under operation '·'. In its least restrictive sense, an abelian group is a set in which one can 'add,' 'subtract,' and 'multiply.'

Fields. A more powerful algebraic structure, known as a field, is a set in which one can 'add,' 'subtract,' 'multiply,' and 'divide.' A commutative ring R with identity is called a field if and only if nonzero elements in R form a group under multiplication. A field with a finite number of elements q is called a finite field or a Galois field (after its discoverer) and is denoted by GF(q). In the sequel p denotes an odd prime and $q = p^m$ denotes a prime power. F_q or GF(q) describes a finite field of q elements; F_q^* describes its group of nonzero elements; and $F_q[x]$ describes the polynomial ring in one indeterminate. The following sections examine the finite computational structure which contains the ring of integer $\mathbf{Z}_M$ with respect to modulus M, the field of integer GF(p) with respect to prime modulus p, and the Galois field GF(q) of p^m elements.

2.3 Modular Arithmetic and Number Theory

This section discusses several basic concepts of modular arithmetic pertaining to the application of number theory to finite computation structures. Number theory for signal processing begins with the elementary idea of division.

Modular arithmetic. Let a and M be any two integers — provided M is positive. Then, a can be divided by M to get a quotient and a remainder; and numbers k and b which satisfy both the equation

$$a = kM + b \tag{2.1}$$

and the inequality

$$0 \leq b < M. \tag{2.2}$$

can be found. The integers a and b are said to be congruent mod M if $a - b \equiv kM$, where k is some integer and M is the modulus. This is written as

$$a \equiv b \bmod M\,. \tag{2.3}$$

Equation (2.3) also serves as the necessary and sufficient condition for the integer a and b to belong to the same residue class mod M. All integers are congruent mod M to some integer in the set $\{ 0, 1, \ldots, M-1 \}$ which is a set of M integers representing all the residue classes mod M. The set is called a residue system mod M or the ring of integers mod M, and is denoted by $\mathbf{Z}_M$. If within a ring of integers the multiplicative inverses a^{-1} exist for all nonzero integers a, that is $aa^{-1} \equiv 1 \bmod M$, then this ring becomes a field. Note that $\mathbf{Z}_M$ is a field if and only if M is a prime. Because of the nature of modular arithmetic, numbers have neither sizes nor magnitudes. For instance, the following basic operations are permissible with modular arithmetic. They are addition (e.g. $15 + 5 = 20 \equiv 3 \bmod 17$); negation (e.g. $-5 = 17 - 5 \equiv 12 \bmod 17$); subtraction (e.g. $15 - 5 = 15 + (-5) \equiv 15 + 12 \bmod 17 \equiv 10 \bmod 17$); multiplication (e.g. $15 \times 5 = 75 \equiv 7 \bmod 17$); inverse (e.g. $7 \equiv 5^{-1} \bmod 17$, because $7 \times 5 \equiv 1 \bmod 17$); and division – provided that a / b exists if and only if b has an inverse, then $a / b = a\, b^{-1}$ (e.g. $15/5 = 15 \times 5^{-1} \equiv 15 \times 7 \bmod 17 \equiv 3 \bmod 17$).

Basis number theory. For a transform having DFT structure, it is necessary to introduce an integer α which represents the Nth root of unity (i.e. $\alpha^N = 1$). The classic study of the problem is developed using Euler's phi-function ϕ which is considered as a function of the integer variable M, denoted as ϕ (M). This function represents the number of integers in $\mathbf{Z}_M$ that are relative primes to M. For M a prime, ϕ $(M) = M - 1$. Furthermore, for M is a power of a prime, p^m, then the only elements of $\mathbf{Z}_M$ which are not relatively prime to p^m are the p^{m-1} multiples of p, therefore ϕ $(p^m) = p^m - p^{m-1} = p^{m-1}(p - 1)$.

For general case that M is a composite number, the fundamental theorem of arithmetic states that M has the following unique prime power factorization

$$M = p_1^{e_1} \cdot p_2^{e_2} \cdots p_n^{e_n}, \tag{2.4}$$

then the general expression for ϕ (M) becomes

$$\phi(M) = M \cdot (1 - 1/p_1) \cdot (1 - 1/p_2) \ . \ . \ . \ (1 - 1/p_n), \tag{2.5}$$

If M is only a prime-factor, which means $e_i = 0$ for $i = 1, 2, \ldots, n$, then

$$\phi(M) = (p_1 - 1) \cdot (p_2 - 1) \ . \ . \ . \ (p_n - 1). \tag{2.6}$$

Euler's theorem states that for every α relatively prime to M

$$\alpha^{\phi(M)} \equiv 1 \bmod M. \tag{2.7}$$

For M prime, this reduces to Fermat's theorem

$$\alpha^{M-1} \equiv 1 \bmod M \tag{2.8}$$

which holds for all nonzero elements of $\mathbf{Z}_M$. In the field $\mathbf{Z}_M$ there are certain roots of unity that are of particular interest. If N is the least positive integer such that

$$\alpha^N \equiv 1 \bmod M, \tag{2.9}$$

then α is said to be a root of unity of order N (or simply of order N) and is denoted as $O_M\alpha = N$. Using another terminology, the root α is said to be a primitive Nth root of unity. For the case of $O_M\alpha = \phi$ (M), α is called a primitive root mod M. If M is prime and α is a primitive root, then all nonzero integers in $\mathbf{Z}_M$ can be generated by powers of α . This characterizes the entire field.

Euler's theorem implies that if α is of order N then N must divide ϕ (M), denoted by $N \mid \phi$ (M). The initial implication suggests that all the possible order of roots are divisors of ϕ (M); however, the theorem does not state that for every divisor of ϕ (M) there are roots corresponding to it. Nevertheless, the phi-function satisfies the property

$$\phi(M) = \phi(N_1) + \phi(N_2) + \ldots + \phi(N_s), \tag{2.10}$$

where N_i is a divisor of M including 1 and M. The following theorem states the relationship of order between roots.

<u>Theorem 2.1</u> If the order of α is N, then the order of α^k is N / gcd(N, k).
Proof: For integers N and k, let $N = mn$, $k = ml$, where n and l are relatively prime. Then, gcd(N, k) = m and lcm(N, k) = nlm. Thus,

$$Nk = m^2nl = \text{gcd}(\ N,\ k\ )\ \text{lcm}(\ N,\ k\ ). \tag{2.11}$$

Assume e is the order of α^k, that is $(\alpha^k)^e = 1$, which indicates e is the smallest positive integer required to make ke a multiple of N. Thus, ke = lcm(N, k). It follows from Equation (2.11) that $ke = Nk$ / gcd(N, k), thus $e = N$ / gcd(N, k). ♦

According to the theorem, if gcd(N, k) = 1 then $O_M\alpha = N$. This implies that the number of roots of order N is given by ϕ (N). Furthermore, the number of primitive roots modulo M is given by ϕ (ϕ (M)) . Overall, Theorem 2.1 allows one to calculate all the roots of possible orders from one of the known primitive roots. To calculate the multiplicative inverse of the nonzero integer α , if M is a prime one finds that $\alpha^{-1} = \alpha^{M-2}$ because $\alpha\ \alpha^{-1} = \alpha^{M-2} = 1$. In general, every nonzero integer α has a multiplicative inverse of the form $\alpha^{-1} = \alpha^{\phi(M)-1}$ with gcd(α , M) = 1. By considering M a composite, one observes that all elements will not have inverses. The elements which are relatively prime to M form an multiplicative group which is a subset of the ring $\mathbf{Z}_M$. The idea presented above is illustrated in Example 2.1.

Reordering powers of a primitive root. This property states that if the first ϕ (N) powers of a primitive root of N are computed modulo N, then all numbers relatively prime to N and less than N are generated. Stated mathematically, let α be a primitive root of N. Then $\alpha^1, \alpha^2, \ldots, \alpha^{\phi(N)}$ are congruent modulo N to $\alpha_1, \alpha_2, \ldots, \alpha_{\phi(N)}$ where $\gcd(\alpha_i, N) = 1$ and $\alpha_i < N$.

Example 2.1 : The ring with arithmetic modulo 15.

Considering $\mathbf{Z}_{15}$ (arithmetic modulo 15), where $M = 15$ is a composite number. Then, $\phi(15) = (5-1)(3-1) = 8$ and $\alpha^{\phi(15)} \equiv 1 \bmod 15$. According to Equations (2, 3), the set of the root of unity is $R = \{ K \mid (K, 15) = 1 \} = \{ 1, 2, 4, 7, 8, 11, 13, 14 \}$. As shown in Table 2.1, R forms a multiplicative group of order 8 which is a subset of the ring $\mathbf{Z}_{15}$.

Table 2.1 The Elements and Their Order of the Multiplicative Group of the Ring $\mathbf{Z}_{15}$

α^1	α^2	α^3	α^4	$N = O_M(\alpha)$	α^{-1}
1	–	–	–	1	1
2	4	8	1	4	8
4	1	–	–	2	4
7	4	13	1	4	13
8	4	2	1	4	2
11	1	–	–	2	11
13	4	7	1	4	7
14	1	–	–	2	14

Table 2.2 shows that there is no primitive root modulo 15, because there is no root having the order of ϕ (15). However, it is important to note that 2, 7, 8, 13 are primitive 4th roots of unity.

♦

Example 2.2 : The field with arithmetic modulo 17.

Considering $\mathbf{Z}_{17}$ (arithmetic modulo 17), where $M = 17$ is prime. According to Equation (9), every nonzero integer α in $\mathbf{Z}_{17}$ has a multiplicative inverse. This means nonzero ele-

Table 2.2 The Possible and Actual Order of the Elements in the Multiplicative Group of the Ring $\mathbf{Z}_{15}$

Possible order N ($N/\phi(M)$)	Number of roots of order N (= $\phi(M)$)	Actual number of roots of N
1	1	1
2	1	3
4	2	4
8	4	0

Note: $15 = \phi(8) + \phi(4) + \phi(2) + \phi(1)$.

ments in $\mathbf{Z}_{17}$ form a group under multiplication. Furthermore, $\mathbf{Z}_{17}$ is also shown to be a commutative ring with unity, such that $\mathbf{Z}_{17}$ is a finite field of order 17 (ϕ (17) + 1 = 16 + 1). By applying the ideas in the previous discussion and referring to Table 2.3, the following observation can be made:

(1) $\phi(17) = 17 - 1 = 16$.

(2) $\alpha^{16} \equiv 1 \bmod 17$, $\gcd(\alpha, 17) = 1$.

(3) The number of the primitive roots is $\phi(\phi(17)) = 8$.

(4) The primitive roots are 3, 5, 6, 7, 10, 11, 12, 14 and also called the primitive 16th roots of unity.

(5) The roots of order N are called primitive Nth roots of unity.

(6) The possible order N of roots, such that $N \backslash \phi(17)$, are 16, 8, 4, 2, 1.

(7) The multiplicative group of nonzero elements in $\mathbf{Z}_{17}$ can be generated by any one of the primitive roots.

(8) For primitive roots α , $(\alpha^8)^2 \equiv 1 \bmod 17$. Thus, $\alpha^8 \equiv -1 \bmod 17$ which becomes $\alpha^3 \equiv 16 \bmod 17$. These primitive roots are also roots of the equation $x^8 + 1 \equiv 0 \bmod 17$.

Table 2.3 The Elements and Their Order of the Field Z_{17}

α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^{15}	α^{16}	N	α^{-1}
0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	?	?
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	1	1
2	4	8	16	15	13	9	1	–	–	–	–	–	–	–	–	8	9
3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1	16	6
4	16	13	1	–	–	–	–	–	–	–	–	–	–	–	–	4	13
5	8	6	13	14	2	10	16	12	9	11	4	3	15	7	1	16	7
6	2	12	4	7	8	14	16	11	15	5	13	10	9	3	1	16	3
7	15	3	4	11	9	12	16	10	2	14	13	6	8	5	1	16	5
8	13	2	16	9	4	15	1	–	–	–	–	–	–	–	–	8	15
9	13	15	16	8	4	2	1	–	–	–	–	–	–	–	–	8	2
10	15	14	4	6	9	5	16	7	2	3	13	11	8	12	1	16	12
11	2	5	4	10	8	3	16	6	15	12	13	7	9	14	1	16	14
12	8	11	13	3	2	7	16	5	9	6	4	14	15	10	1	16	10
13	16	4	1	–	–	–	–	–	–	–	–	–	–	–	–	4	4
14	9	7	13	12	15	6	16	3	8	10	4	5	2	11	1	16	11
15	4	9	16	2	13	8	1	–	–	–	–	–	–	–	–	8	8
16	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2	16

(9) For roots of order 4 which are 4 and 13, $(\alpha^2)^2 \equiv 1 \mod 17$, therefore, $\alpha^2 \equiv -1$ mod 17 and thus, $\alpha^2 \equiv 16 \mod 17$. The root α is also a root of the equation $x^2 + 1 \equiv 0 \mod 17$. By definition if the congruence $x^2 \equiv c \mod M$, where $(c, M) = 1$, is solvable, then c is a quadratic residue modulo M. If this congruence has no solution, c is said to be a quadratic nonresidue modulo M. For instance, -1 is a quadratic residue modulo 17 and 2 is also a quadratic residue modulo 17 — in the sense of $11^2 \equiv 2 \mod 17$.

(10) To find all the primitive roots, one simply has to find all k with $(k, 16) = 1$. By raising a known primitive root to power k, all primitive roots can be found. In the latter example, 3 is a primitive root modulo 17 and $k = \{ 1, 3, 5, 7, 9, 11, 13,$

15 }. The set A of the primitive roots is $A = \{ 3^1, 3^3, \ldots, 3^{15} \} = \{ 3, 10, 5, \ldots,$ $6 \}$.

(11) Given the fact that $M = 17 = 2^b + 1$ with $b = 4$, and $2^b \equiv -1 \bmod 17$. This makes $(2^b)^2 \equiv 1 \bmod 17$ such $(2^b)^2 \equiv 1 \bmod 17$ such that integer 2 has order $2b$. ♦

2.3.1 Chinese Remainder Theorem (CRT) for Integers

A special case of this theorem is credited to the Chinese mathematician Sun-Tsu, who wrote sometime between 200 B.C. and 200 A.D.. A general proof given by Chiu-Shao, Nicomachus (Greek), and Euler. When considering a nonprime M, $\mathbf{Z}_M$ is a ring and the inverses exist only for integers relatively prime to M. Let M have the prime power factorization found in Equation (2.4). When the arithmetic is performed in modulo M, it is in effect performed modulo for each prime power $p_i^{e_i}$ simultaneously. A set of arithmetic operations can be performed modulo for each $p_i^{e_i}$ separately with the final result mod M obtained using the Chinese remainder theorem (CRT). The general CRT theorem for integers states as follows:

Given primes $p_1, p_2, \ldots, p_n$ and integers $c_1, c_2, \ldots, c_n$, the simultaneous congruence $C \equiv c_i \bmod p_i^{e_i}$ have a unique solution mod M, and

$$C = \left(\sum_{i=1}^{n} c_i N_i M_i \right) \bmod M \tag{2.12}$$

where $M_i = M / p_i^{e_i}$, with g.c.d$(M_i, p_i^{e_i}) = 1$, and $N_i \equiv (M_i)^{-1} \bmod p_i^{e_i}$.

2.3.2 Residue Number System Arithmetic (RNS)

Let a and b be determined by the CRT from the sequences of integers $\{ a_i \}$ and $\{ b_i \}$, $i = 1, 2, \ldots, L$, respectively (i.e., $a_i \equiv a \bmod N_i$ and $b_i \equiv b \bmod N_i$). Let $\otimes$ denote the operation of either '$\cdot$ 'or '+'. Then it is easy to show that

$$(a \otimes \mathrm{b}) \bmod N = \{ (a_1 \otimes b_1) \bmod N_1, \ldots, (a_L \otimes b_L) \bmod N_L \},$$

where $N = N_1 N_2 \cdots N_L$. Thus multiplication, addition, and subtraction involving a and b can be accomplished solely by operations on the residue digits a_i and b_i . Such arithmetic is called residue number system (RNS) arithmetic. High speed digital systems can be mechanized by parallel processors operating on the residue digits. As in any digital system, there are overflow constraints.

Modulo techniques for binary systems. Closely related to the concept of congruence is the idea of extracting a residue. Computations involving residues are usually simple because one never needs to work with quantities larger than the modulus. Two procedures which simplify some of the operations are (1) $x \pm y \bmod M \equiv [\,(x \bmod M) \pm (y \bmod M)\,] \bmod M$; and (2) $x \cdot y \bmod M \equiv [\,(x \bmod M) \cdot (y \bmod M)\,] \bmod M$.
One method of computing '$a \bmod M$' is simply to divide a by M and keep the remainder. If M is a power of two, and a is represented on a binary machine, then a trivial method of extracting $a \bmod M$ exists:

$$a \bmod 2^k = (\sum_i a_i.2^i) \bmod 2^k = \sum_{i=0}^{k-1} a_i.2^i \,. \tag{2.13}$$

This operation is performed by masking out all but the k least significant bits. Another simple case occurs when the modulus is of the form $2^k - 1$. Let the first k bits of a, $a_1, a_2, \ldots, a_{k-1}$, be the binary number A; let the next k bits be the binary number

$$B = a_k + 2a_{k+1} + 4a_{k+2} + \ldots$$

and so on, then

$$a = A + 2^k B + 2^{2k} C + \ldots . \tag{2.14}$$

Since $2^k \bmod (2^k - 1) \equiv 1$, and $2^{nk} \bmod (2^k - 1) \equiv 1$, thus

$$a \bmod (2^k - 1) \equiv (A + B + C + \ldots) \bmod 2^k - 1 \,. \tag{2.15}$$

Consequently, the residue of a number with very many bits may be found by adding k-bit subwords.

Another case in which a residue is easily extracted without division is when the modulus is of the form $2^k + 1$. Let a be as in (2.14), then since $2^k \equiv -1 \bmod (2^k + 1)$, and $2^{nk} \equiv (-1)^n \bmod (2^k + 1)$, then

$$a \bmod (2^k + 1) \equiv (A - B + C - \ldots) \bmod 2^k + 1 .$$

Thus we have seen that for moduli of the form 2^k, $2^k \pm 1$, methods much simpler than division exist for extracting a residue.

2.4 Residue Polynomials

Polynomials with coefficients in a field (ring) are referred to as polynomials over a field (ring) and are important in the development of efficient cyclic convolution evaluation. This section discusses properties of polynomials. Many properties are analogous to the properties of integers discussed in the previous section. For example, the CRT for polynomials is similar to that for integers and results in an expansion that reduces multiplications in FFT implementations.

2.4.1 Properties of Polynomial

The theory of residue polynomials is closely related to the theory of integer residue classes. For each field F, there is a ring $F[x]$ called the ring of polynomials over F. Within a ring of polynomials, subtraction is always possible, but division is not. We write $f(x) \mid g(x)$ and say that the polynomial $g(x)$ is divisible by $f(x)$ — provided there is a polynomial $q(x)$ such that $g(x) = q(x) \cdot f(x)$. A nonzero polynomial $f(x)$ that is divisible only by $f(x)$ or α, where α is an arbitrary field element, is called an irreducible polynomial. In the case that $f(x)$ is not a divisor of $g(x)$, the division of $g(x)$ by $f(x)$ will produce a quotient and residue polynomial, such that

$$g(x) = q(x) \cdot f(x) + r(x), \tag{2.16}$$

where the degree of $r(x)$ is less than the degree of $f(x)$. Usually, the residue polynomial is of more interest than the quotient polynomial. The residue polynomial in congruence form is written as

$$r(x) \equiv g(x) \bmod f(x). \tag{2.17}$$

The process of obtaining the remainder $r(x)$ from $g(x)$ and $f(x)$ is called polynomial residue reduction. In general all polynomials of the same residue, when divided by $f(x)$, are said to be congruence modulo $f(z)$ and the relation is denoted by

$$g_i(x) \equiv r_i(x) \bmod f(x). \tag{2.18}$$

Two polynomials which differ only by a multiplicative constant are congruent. Thus, residue polynomials deal with the relative values of coefficients rather than with their absolute values. At this point, it is worth noting that when dealing with polynomials, primary interest lies in the coefficients of the polynomial. Consequently, a set of N elements $a_0, a_1, \ldots, a_{N-1}$ is arranged in the form of a polynomial

$$h(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{N-1}x^{N-1} \tag{2.19}$$

with x denoting position. This feature is very important in digital signal processing because each polynomial coefficient represents a sample of an analog signal stream which defines its location and intensity.

Equation (2.18) defines equivalence classes of polynomials modulo a polynomial $f(x)$. It is easily verified that the set of polynomials defined with addition and multiplication modulo $f(x)$ is a ring and reduces to a field when $f(x)$ is irreducible. When $f(x)$ is not irreducible, it can always be factored uniquely into powers of irreducible polynomials. Note, however, that the factorization depends on the field of its coefficients: $x^2 + 1$ is irreducible for coefficients in the field of rational numbers, but it is reducible in the finite field $\mathbf{Z}_{17}$ where $x^2 + 1 = (x-4)(x-13)$, as well as in the field of complex numbers where $x^2 + 1 = (x+i)(x-i)$, $i = \sqrt{-1}$.

The CRT for polynomials. Now suppose that $f(x)$ is the product of n polynomials $f_i(x)$ having no common factors, then these polynomials are usually called relatively prime polynomials.

$$f(x) = \prod_{i=1}^{n} f_i(x)^{e_i} . \tag{2.20}$$

Since each of these polynomials $f_i(x)$ is relatively prime with all other polynomials, it has an inverse modulo every other polynomial. This means that the CRT can be extended to the ring of polynomials modulo $f(x)$ which allows the unique expression of $g(x)$ as a function of polynomials $g_i(x)$ obtained by reducing the $g(x)$ modulo to the polynomials $f_i(x)$. The CRT for polynomials is then expressed as

$$g(x) \equiv (\sum_{i=1}^{n} g_i(x)\ N_i(x)\ M_i(x)) \bmod f(x) ,$$

where $M_i(x) = f(x)/f_i(x)^{e_i}$, with gcd($M_i(x)$, $f_i(x)^{e_i}$) = 1 and $N_i(x) \equiv M_i^{-1}(x) \bmod f_i(x)^{e_i}$. The most difficult part of the calculation of $N_i(x)M_i(x)$ relates to the evaluation of the inverse of $N_i(x)$ which can be accomplished using Euclid's algorithm.

2.4.2 Convolutions and Polynomial RNS

Polynomial algebra plays an important role in digital signal processing because convolutions and discrete Fourier transforms (DFTs) can be expressed in terms of operations on polynomials. This is seen by considering the simple convolution g_l of two sequences h_n and h'_m of N terms

$$g_l = \sum_{n=0}^{N-1} h_n h'_{l-n} \tag{2.21}$$

where $l = 0, 1, \ldots, 2N-2$. Suppose that the N elements of h_n and h'_m are assigned to be the coefficients of polynomials $h(x)$ and $h'(x)$ of degree $N-1$ in x, where x is the polynomial variable. Analogous to Equation (2.19), if $h(x)$ is multiplied by $h'(x)$ the resulting polynomial $g(x)$ will be of degree $2N-2$. Thus,

$$g(x) = h(x)h'(x) = \sum_{l=0}^{2N-2} a_l x^l . \tag{2.22}$$

With the polynomial multiplication, each coefficient a_l of x^l is obtained by summing all products $h_n h'_m$ such that $n + m = l$. It follows from Equation (2.21) and the fact that $m = l - n$ that

$$a_l = g_l = \sum_{n=0}^{N-1} h_n h'_{l-n} . \tag{2.23}$$

Thus, Equation (2.22) becomes

$$g(x) = \sum_{l=0}^{2N-2} g_l x^l . \tag{2.24}$$

This means that the convolution of two sequences can be treated as the multiplication of two polynomials. Moreover, if the convolution is cyclic, the indices l, m, and n are defined modulo N. Thus in N-term cyclic convolutions, $N \equiv 0 \bmod N$ which implies that $x^N \equiv 1$. Therefore a cyclic convolution can be viewed as the product of two polynomials modulo the polynomial $f(x) = x^N - 1$, namely

$$g(x) = h(x)h'(x) \bmod f(x) . \tag{2.25}$$

According to the previous discussion of CRT for polynomials, the polynomial $g(x)$ is first computed using the reduced polynomial equations, $h_i(x) \equiv h(x) \bmod f_i(x)$ and $h'_i(x) \equiv h'(x) \bmod f_i(x)$, where $f_i(x)$s are factors of $x^N - 1$; then, the n polynomial products are evaluated

$$g_i(x) \equiv h_i(x)h'_i(x) \bmod f_i(x) \tag{2.26}$$

with $g(x)$ reconstructed from $g_i(x)$ using the CRT.

Up to this point, the problem of computing the Equation (2.25) can be simplified somewhat by applying the CRT — provided the polynomial $f(x)$ can be factored as

$$f(x) = \prod_{l=0}^{n} f_l(x) \; . \qquad (2.27)$$

In accordance with Winograd's theorem [Nus82], a convolution algorithm can be synthesized with $2N - k$ multiplications where k is the number of irreducible factors of $f(x)$ over the field F. If F is the field of complex numbers **C**, then $x^N - 1$ factors into N polynomials $x - \omega^i$ of degree 1 where $\omega = \exp(-j2\pi/N)$ and $j = \sqrt{-1}$. In this case, the computation technique defined by Winograd is equivalent to the DFT approach and requires only N general multiplications. Unfortunately, the roots ω are irrational and complex so that the multiplications by scalars corresponding to DFT computation must also be considered as general complex multiplications.

Primitive roots of unity. The element ω^m is a primitive root of unity if the set { $(\omega^m)^0$, $(\omega^m)^1 , \ldots , (\omega^m)^{N-1}$ } can be reordered as { $\omega^0 , \omega^1 , \ldots , \omega^{N-1}$ }. Drawing the unit circle tn the complex plane and showing the points $\omega^0 , \omega^1 , \ldots , \omega^{N-1}$ verifies that ω^m is a primitive root if and only if gcd(m, N) = 1.

Factorization of $x^N - 1$. When F is the field of rational numbers **Q**, the polynomial $x^N - 1$ factors into polynomials having rational number coefficients. These polynomials are called cyclotomic polynomials and are irreducible for coefficients in the field of rational numbers,

$$z^N - 1 = \prod_{i/N} C_i(z)$$

where $C_i(z)$ is a cyclotomic polynomial of index i and the values of i used for $i \,/\, N$ include 1 and N. For example, $z^2 - 1 = (z-1)(z+1) = C_1(z)\, C_2(z)$ and $z^4 - 1 = (z-1)(z+1)(\, z^2 + 1) =$

$C_1(z)\,C_2(z)\,C_4(z)$. Thus, one cyclotomic polynomial of degree n_i exists for each divisor N_i of N and n_i is shown to be $\phi(N_i)$.

Example 2.3 : Polynomial factorization over fields.

Let $f(x) = x^5 - 1$, then $f(x)$ is factored over the complex field **C** as

$$f(x) = \prod_{i=0}^{4}(x - \omega_5^i),$$

over the real field **R** as

$$f(x) = (x-1)\ (x^2 + 2\cos(2\pi/5)x + 1)\ (x^2 + 2\cos(4\pi/5)x + 1),$$

and over the rational field **Q** as

$$f(x) = (x-1)\ (x^4 + x^3 + x^2 + x + 1).\ \blacklozenge$$

Factorization of $x^N - 1$ over finite fields. In order to obtain the maximum k value, the polynomial factorization over a finite field must be considered. Let $f(x)$ be an integral polynomial (i.e. with integer coefficients) of degree m, and M be a natural number. If c is an integer such that $f(c)$ is divisible by M, then c is a solution of the algebraic congruence

$$f(x) \equiv 0 \bmod M, \tag{2.28}$$

and all values x for which $x \equiv c \bmod M$ are also solutions. All the solutions belonging to the same residue class modulo M as c are considered as a single solution. Therefore to determine all the solutions of the congruence in Equation (2.28), one need only try the values $x = 0, 1, 2, \ldots, M-1$. It is known that [Lip81] the algebraic congruence of degree m in Equation (2.28), if M is prime, has at most m incongruent solutions modulo M. According to Equations (2.8) and (2.9), it follows that the congruence

$$x^{M-1} - 1 \equiv 0 \bmod M$$

has roots $x = 1, 2, 3, \ldots, M-1$. This means $f(x) = x^{M-1} - 1$ can be factored over the finite field $\mathbf{Z}_M$ as

$$x^{M-1} - 1 \equiv \prod_{i=0}^{M-2} (x - \alpha^i) \bmod M \; ,$$

where α is one of the prime root mod M.

2.5 Finite Fields Based on Polynomial Rings

The field of integers modulo a prime number is the most familiar example of a finite field, but many of its properties extend to arbitrary finite fields. The characterization of finite fields shows that every finite field is of prime-power order and that, conversely, for every prime power there exists a finite field whose number of elements is exactly that prime-power. Furthermore, finite fields with the same number of elements are isomorphic and may therefore be identified. The parallel between the ring of integers and the ring of polynomials over a field must be apparent. Both are special cases of an algebraic structure called a Euclidean ring.

2.5.1 Construction of the Finite Field GF(p^m)

Let $f(x)$ be a polynomial of degree m in $F_p[x]$, then by the previous discussed Euclidean division algorithm and the technique of polynomial residue reduction the rules of composition in $F_p[x]$ are

1) Addition

$$a(x) + b(x) = g_1(x) \equiv r_1(x) \bmod f(x) \; ,$$

where $g_1(x) = q_1(x)f(x) + r_1(x)$.

2) Multiplication

$$a(x) \; . \; b(x) = g_2(x) \equiv r_2(x) \bmod f(x) \, ,$$

where $g_2(x) = q_2(x)f(x) + r_2(x)$.

Consider the set G of all polynomials of degree less than m, it is known [Mac77] that the set G is of order p^m and forms an abelian group with respect to modulo $f(x)$ addition. Furthermore, if $f(x)$ is irreducible, all nonzero elements of G have multiplicative inverses. Hence, the set G forms a field with respect to modulo $f(x)$ arithmetic. In generating fields using modulo $f(x)$ arithmetic, the field F_p from which the coefficients of the polynomial are chosen is called a subfield of G. The field G is called the extension field of degree m over F_p and is denoted by F_{p^m} or GF(p^m). On the other hand, all the multiples of the polynomial $f(x)$ form an ideal which is denoted by ($f(x)$). By applying residue reduction to this ideal, a residue class ring known as the ring of polynomials modulo $f(x)$ is formed and is denoted by $F_p[x]/(f(x))$. It is a well established fact that an isomorphism between the extension field F_{p^m} and the residue class ring $F_p[x]/(f(x))$ exists. The extension field F_{p^m} may also be viewed as a vector space over F_p. This view results from the following properties.

The elements of F_p form an abelian group under addition. Moreover, each "vector" $\beta \in F_{p^m}$ can be multiplied by a "scalar" $b \in F_p$ such that $b\beta \in F_{p^m}$. The laws for scalar multiplications are also satisfied:

$$b(\beta_1 + \beta_2) = b\beta_1 + b\beta_2,$$

$$(a + b)\beta_1 = a\beta_1 + b\beta_1, \text{ and}$$

$$1 \cdot \beta = \beta$$

where $a, b \in F_p$, and $\beta_1, \beta_2, \beta \in F_{p^m}$. Another way of viewing the extension concept of extension fields is based on the fact that the irreducible polynomial $f(x)$ over F_p has a root in a field F_{p^m}. Hence, it is said that the smallest extension field F_{p^m} is obtained by adjoining a root α of $f(x)$ to the ground field F_p and is denoted by $F_p(\alpha)$. The vector form representation of this field is

$$F_{p^m} = F_p(\alpha) = \{ b_0 + b_1\alpha + b_2\alpha^2 + \dots + b_{n-1}\alpha^{m-1} : \alpha \in F_{p^m}, b_i \in F_p \}. \quad (2.29)$$

One of the important properties of the additive structure of a finite field is illustrated as follows. Let F_r be an arbitrary finite field of order r. The field contains the unit element 1 and since the field is finite which means that the elements $1, 1 + 1 = 2, 1 + 1 + 1 = 3, \ldots$ can not all be distinct. Therefore, there exists a smallest number p such that $p = 1 + 1 + \ldots + 1$ (p times) = 0. This p must be a prime number (for if $rs = 0$ then $r = 0$ or $s = 0$) and is called the characteristic of the field. Thus for $r = q^n$, $q = p^m$ where p is prime, one concludes that F_r is an extension field of degree n over F_q, which is a field of degree m over the prime subfield F_p ; and all these fields have the same characteristic p.

A similar analysis shows that for any nonzero element $\alpha \in F_r$ there is a positive integer t such that $\alpha^t = 1$, the least such t being called the order of α. Moreover, the integer t is a divisor of $r - 1$ which is the order of the multiplicative group of F_r ; and so, α satisfies $\alpha^{r-1} = 1$. In the case of $t = r - 1$, such elements are called primitive elements of F_r . The most prominent feature of finite fields concerns their cyclic multiplicative groups. The elements of F_r in power form are written as

$$F_r = \{ 0, 1, \alpha, \alpha^2, \ldots, \alpha^{r-2} \} . \tag{2.30}$$

It is convenient to introduce a formal symbol $-\infty$ defined by the equation $\alpha^{-\infty} = 0$, so that the general elements of F_r can be expressed in power form. If the element $\beta = x$ is a primitive element of the field generated by $f(x)$, then $f(x)$ is called a primitive polynomial. The following theorem states the property of a primitive polynomial.

<u>Theorem 2.2</u> If $f(x)$ is a primitive polynomial, then $s = r - 1$ is the smallest integer such that $f(x)$ divides $x^s - 1$.

Proof. If $f(x)$ divides $x^s - 1$, then $x^s - 1 = f(x)\, g(x)$. That is $x^s - 1 \equiv 0 \bmod f(x)$, hence $x^s \equiv 1 \bmod f(x)$. Since $f(x)$ is a primitive polynomial then x is a primitive element of the field F_r , thus $s = p^m - 1$ is the order of x and also the smallest integer such that $x^s \equiv 1 \bmod f(x)$. ♦

Example 2.4 : The representations of the elements in the field F_{16} (or GF(2^4)).

To represent the elements of F_{16} in this way, regard it as a simple algebraic extension of F_2 of degree 4 which is obtained by adjoining a root α of an irreducible polynomial $f(x)$ over F_2. Consider defining irreducible polynomials as primitive and non-primitive. Let $f_1(x) = x^4 + x + 1$ and $f_2(x) = x^4 + x^3 + x^2 + x + 1$. Because $x^{15} \equiv 1 \bmod f_1(x)$, that $\alpha = x$ is a primitive element of the finite field $F_2[x]/(f_1(x))$ and the polynomial $f_1(x)$ is a primitive polynomial. From the fact that $x^5 \equiv 1 \bmod f_2(x)$, $\alpha = x$ is thus an element of order 5. This implies that $f_2(x)$ is not a primitive polynomial. However, $\alpha = x + 1$ is shown to be a primitive element of $F_2[x]/(f_2(x))$. Accordingly, the isomorphic relationships within finite fields conclude that $F_{16} \cong F_2[x]/(f_1(x)) \cong F_2[x]/(f_2(x))$. The elements of the fields represented in power form and vector form are shown in Table 2.4 and Table 2.5. ♦

Table 2.4 Elements $\beta = a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0$ of $F_2(\alpha)$ Generated by $f_1(x)$.

β	$\phi(\beta)$	a_3	a_2	a_1	a_0	β	$\phi(\beta)$	a_3	a_2	a_1	a_0
$\alpha^{-\infty}$	0	0	0	0	0	α^7	15	1	0	1	1
α^0	1	0	0	0	1	α^8	15	0	1	0	1
α	15	0	0	1	0	α^9	5	1	0	1	0
α^2	15	0	1	0	0	α^{10}	3	0	1	1	1
α^3	5	1	0	0	0	α^{11}	15	1	1	1	0
α^4	15	0	0	1	1	α^{12}	5	1	1	1	1
α^5	3	0	1	1	0	α^{13}	15	1	1	0	1
α^6	5	1	1	0	0	α^{14}	15	1	0	0	1

Table 2.5 Elements $\beta = a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0$ of $F_2(\alpha)$ Generated by $f_2(x)$.

β	$\phi(\beta)$	a_3	a_2	a_1	a_0	β	$\phi(\beta)$	a_3	a_2	a_1	a_0
$\alpha^{-\infty}$	0	0	0	0	0	α^7	15	0	1	1	1
α^0	1	0	0	0	1	α^8	15	1	0	0	1
α	15	1	1	0	0	α^9	5	0	1	0	0
α^2	15	0	1	0	1	α^{10}	3	1	1	0	0
α^3	5	1	1	1	1	α^{11}	15	1	0	1	1
α^4	15	1	1	1	0	α^{12}	5	0	0	1	0
α^5	3	1	1	0	1	α^{13}	15	0	1	1	0
α^6	5	1	0	0	0	α^{14}	15	1	0	1	0

2.5.2 Properties of the Finite Field GF(p^m)

A useful property of the finite field of prime characteristic is described by Theorem 2.3.

Theorem 2.3 In a field of characteristic p, $(\alpha + \beta)^p = \alpha^p + \beta^p$ for $\alpha, \beta \in F$.

Proof. By the binomial theorem

$$(\alpha + \beta)^p = \alpha^p + \binom{p}{1}\alpha^{p-1}\beta + \ldots + \binom{p}{p-1}\alpha\beta^{p-1} + \beta^p, \quad (2.31)$$

claiming that for $1 \le k \le p-1$, the binomial coefficient

$$\binom{p}{k} = \frac{p(p-1)(p-2)\ \ldots\ (p-k+1)}{k!} \quad (2.32)$$

has p as a factor. First we note that $k!$ divides $p(p-1)\cdot\cdot\cdot(p-k+1)$, since the binomial coefficient are integers. But $(k!, p) = 1$, since p is prime. Consequently, Equation (2.32) becomes

$$\binom{p}{k} = pm \tag{2.33}$$

for some integer m. Note that for any x in a domain of characteristic p, $mpx = 0$. Thus, every term on the right-hand side of Equation (2.31) vanishes except α^p and β^p. ♦

As an immediate corollary, the following relationship holds. $(\alpha_0 + \alpha_1 + \ldots + \alpha_s)^{p^n} = \alpha_0^{p^n} + \alpha_1^{p^n} + \ldots + \alpha_s^{p^n}$, for $\alpha_i \in F$, $i = 0, 1, \ldots, s$, and $n \in N$.

Now consider further details on the properties of finite fields such as minimal polynomials, conjugates, and automorphisms. According to Theorem 2.3, $(\alpha + \beta)^p = \alpha^p + \beta^p$. Since $0^p = 0$, $1^p = 1$, and $(\alpha\beta)^p = \alpha^p\beta^p$, it follows that the operation of taking pth powers preserves all the structure of the finite field GF(p^m). Plainly, the p^i th power function will also define an automorphism of GF(p^m) for each i. Fermat's theorem implies that every element β of GF(p^m) satisfies the equation $x^{p^m} - x = 0$. Where this polynomial has all its coefficients from the subfield GF(p) and is monic, β may satisfy a lower degree equation. A polynomial ,$\Phi(x)$, of the smallest degree with coefficients in GF(p) such that $\Phi(\beta) = 0$ is called the minimal polynomial of β over GF(p) and is denoted by $\Phi_\beta(x)$. The following theorems illustrate properties of minimal polynomial.

<u>Theorem 2.4:</u> Let $f(x)$ be a polynomial of degree l with coefficients in GF(p), which is irreducible in this field, and let β be a root of $f(x)$ in an extension field. Then $\beta, \beta^p, \beta^{p^2}, \ldots, \beta^{p^{l-1}}$ are all the roots of $f(x)$.

Proof. Let $f(x) = a_0 + a_1x + \ldots + a_lx^l$, then by Theorem 2.3 $(f(x))^p = \alpha_0^p + \alpha_1^p x^p + \ldots + \alpha_l^p x^{lp} = a_0 + a_1(x^p) + \ldots + a_l(x^p)^l = f(x^p)$. If $f(\beta) = 0$, then $f(\beta^p) = (f(\beta))^p = 0$. By induction, one finds that $\beta^{p^2}, \ldots, \beta^{p^{l-1}}$ are roots of $f(x)$. The following argument shows that these l field elements are distinct. Suppose the field elements are not distinct, and $\beta^{p^i} = \beta^{p^j}$, and suppose $j < i$. Then $\beta = \beta^{p^l} = (\beta^{p^i})^{p^{l-i}} = (\beta^{p^j})^{p^{l-i}} = (\beta)^{p^{l+j-i}}$, therefore $1 = \beta^{(p^{l+j-i}-1)}$. Thus, the order of β

divides $p^{l+j-i} - 1$. Since $f(x)$ differs from the minimal polynomial of β by at most a constant factor and $l + j - i < l$, this contradicts the definition of minimal polynomial. Therefore β, β^p, $\beta^{p^2}, \dots, \beta^{p^{l-1}}$ are distinct. Since $f(x)$ can have at most l roots, these must be all the roots. ♦

Theorem 2.5: Let $f(x)$ be an irreducible monic polynomial with coefficient from GF(p) for which $f(\beta) = 0$, where β is an element of some extension field of GF(p). Then, $f(x) = \Phi_\beta(x)$.

Proof. Let $f(x) = q(x)\Phi_\beta(x) + r(x)$. Since $\Phi_\beta(\beta) = 0$, and $f(\beta) = q(\beta)\Phi_\beta(\beta) + r(\beta) = 0$, then $r(\beta) = 0$. This implies $\Phi_\beta(x)$ divides $f(x)$, however $f(x)$ is irreducible, such that $f(x) = \Phi_\beta(x)$. ♦

Theorem 2.6: All the roots of an irreducible polynomial have the same order.

Proof. Given by Peterson[Pet72a]. ♦

Theorem 2.7: If $f(x)$ is irreducible with β as a root then deg($f(x)$) is the smallest l such that $\beta^{p^l} = \beta$.

Proof. Given by Peterson[Pet72a]. ♦

Example 2.5 Calculation of the minimal polynomials of β in GF(2^4).

Consider the field GF(2^4) formed by modulo $f(x)$ arithmetic with $f(x) = x^4 + x + 1$. Find the minimal polynomial $\Phi_\beta(x)$ of β in GF(2^4). From Theorem 2.4, one has $\Phi_\beta(x) = (x - \beta)(x - \beta^p) \dots (x - \beta^{p^{m-1}})$. Thus, the minimal polynomials of β in GF(2^4) are listed as follows:

$$\Phi_\alpha(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8);$$

$$\begin{aligned}\Phi_{\alpha^3}(x) &= (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^{24}) \\ &= (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^9);\end{aligned}$$

$$\Phi_{\alpha^5}(x) = (x - \alpha^5)(x - \alpha^{10}); \text{ and}$$

$$\begin{aligned}\Phi_{\alpha^7}(x) &= (x - \alpha^7)(x - \alpha^{14})(x - \alpha^{28})(x - \alpha^{56}) \\ &= (x - \alpha^7)(x - \alpha^{14})(x - \alpha^{13})(x - \alpha^{11}).\end{aligned}$$

To solve the coefficients of the minimal polynomial, start evaluating $\Phi_\alpha(x)$. Assume $\Phi_\alpha(x)$ $= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4$. Because $\Phi_\alpha(x)$ is monic and irreducible, thus $a_4 = 1$, $a_0 = 1$.

From $\Phi_\alpha(\alpha) = 0$, one finds that

$$1 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + \alpha^4 = 0\,,$$

$$1 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + (\alpha + 1) = 0\,, \text{ and}$$

$$(a_1 + 1)\alpha + a_2\alpha^2 + a_3\alpha^3 = 0\,.$$

These indicate that $a_1 = 1$, $a_2 = 0$, and $a_3 = 0$. Thus $\Phi_\alpha(x) = x^4 + x + 1$. In the case of $\Phi_{\alpha^3}(x)$, let $\Phi_{\alpha^3}(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4$. Similarly $a_4 = 1$, $a_0 = 1$, and $1 + a_1\alpha^3 + a_2\alpha^6 + a_3\alpha^9 + \alpha^{12} = 0$. From Table 2.4, it is found that $\alpha^6 = \alpha^3 + \alpha^2$, $\alpha^9 = \alpha^3 + \alpha$, and $\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$; such that Equation (2.8) becomes

$$1 + a_1\alpha^3 + a_2(\alpha^3 + \alpha^2) + a_3(\alpha^3 + \alpha) + (\alpha^3 + \alpha^2 + \alpha + 1) = 0$$

$$(a_1 + a_2 + a_3 + 1)\alpha^3 + (a_2 + 1)\alpha^2 + (a_3 + 1)\alpha = 0$$

which implies $a_3 = 1$, $a_2 = 1$, $a_1 = 1$ and $\Phi_{\alpha^3}(x) = 1 + x + x^2 + x^3 + x^4$. ♦

2.5.3 Bases of GF(p^m) over GF(p)

The finite field GF(p^m) can be regarded as a vector space of dimension m over GF(p). Any set of m linearly independent elements can be used as a basis for this vector space. Thus, the number of distinct bases is usually rather large [Lid86]. Accordingly, it is wise to restrict one's attention to certain special types of basis. Different vector basis representations of field elements result in different architecture of field arithmetic system. Due to their respective distinct features that make them suitable for specific applications, this research effort investigates three basis types of particular importance. They are primal basis (or standard basis), normal basis, and dual basis (or complementary basis). A system based on primal bases does not require basis conversion, hence it is readily matched to any input or output system. The normal basis system is effective in performing operations such as finding inverse element. performing the squaring (in GF(2)) or exponentiation of a field element. A dual basis system can easily perform multiplication of a field element x and the power of a primitive element α . A detailed discussion involving the architectural designs stemming from these bases follow later in the chapter.

It is important to introduce the concept of the trace of a field element which is a very useful analytic tool in finite fields. Let α be an element of GF(p^m), then its trace function Tr(α) over GF(p) is defined as

$$\mathrm{Tr}(\alpha) = \alpha + \alpha^p + \alpha^{p^2} + \dots + \alpha^{p^{m-1}} . \tag{2.34}$$

The trace function satisfies the following properties (shown without proofs):

For all α , $\beta \in$ GF(p^m),

1) Tr(α) $\in$ GF(p), the function is a linear transformation from GF(p^m) onto GF(p);

2) Tr(α +β) = Tr(α) + Tr(β);

3) Tr($\lambda \cdot \alpha$) = $\lambda \cdot$ Tr(α), if $\lambda \in$ GF(p);

4) Tr(α)^p = Tr(α^p) = Tr(α); and

5) Tr(a) = na, $a \in$ GF(p).

Primal basis. The most popular basis is primal basis

$$N_p = \{ 1, \alpha, \alpha^2, \dots , \alpha^{n-1} \} \tag{2.35}$$

consists of the powers of a defining element α of GF(p^m) over GF(p).

Dual basis. The second type of basis, dual basis, is defined that bases

$$N_\alpha = \{ \alpha_0, \alpha_1, \dots , \alpha_{n-1} \} \tag{2.36}$$

and

$$N_\beta = \{ \beta_0, \beta_1, \ldots, \beta_{n-1} \} \tag{2.37}$$

of GF(p^m) over GF(p) are dual bases if for $1 \le i,\ j \le n-1$, one has $\mathrm{Tr}(\alpha_i\beta_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta function, equal to 1 if i is equal to j and zero otherwise.

Normal basis. The last type of basis is normal basis

$$N_n = \{ \alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{n-1}} \} \tag{2.38}$$

which is defined by a suitable primitive element α of GF(p^m).

Example 2.6 The bases in GF(2^4) over GF(2).

Continuing Example 2.4, let $\alpha \in$ GF(2^4) be a primitive root of the irreducible polynomial $f_1(x)$ in $F_2[x]$. Then, $N_p = \{ 1, \alpha, \alpha^2, \alpha^3 \}$ is a primal basis over GF(2^4). Its uniquely determined dual basis is easily checked, which is $N_d = \{ 1+\alpha^3, \alpha^2, \alpha, 1 \}$. The basis $N = \{ \alpha, \alpha^2, \alpha^4, \alpha^8 \}$ is not a normal basis because $\alpha + \alpha^2 + \alpha^4 + \alpha^8 = 0$ such that N is not linearly independent. However, a normal basis can be found by choosing another primitive root α^3 and raising it to the powers of 2 such that $N_n = \{ \alpha^3, \alpha^6, \alpha^{12}, \alpha^{24} \} = \{ \alpha^3, \alpha^6, \alpha^{12}, \alpha^9 \}$. ♦

CHAPTER 3
COMPUTATIONS IN FINITE FIELDS

3.1 Introduction

For obvious reasons the arithmetic over finite field GF(q) has to be distinguished between the cases that: 1) The order of the field is a prime that $q = p$, then arithmetic in GF(q) is that of $\mathbf{Z}_p$, the field of integers mod p; 2) The order of the field is a prime power that $q = p^m$, $m > 1$, then the arithmetic in GF(q) becomes that of an m-dimension algebra over GF(p). In the later case, the complexity of system structure becomes more involved on the size of the field and also on the data structure which is used to represent elements of the field.

In the previous process of constructing GF(p^m) from GF(p), two representations – the power form and the polynomial form – for the nonzero elements of GF(q) were developed. In this chapter, the structures and applications of these representations are investigated. Zech's logarithm [Con68] solves the problem of the nontrivial addition operation of power form representation in finite fields. However, as the size of fields becomes larger, the conversion between power and polynomial forms shows to be a challenge task which is traditionally termed as the discrete logarithm and exponentiation problem, and treated in separate as an interest research topic in the area of cryptography. Hence, it is not proper to conduct arithmetic operation in the power form when the field has large order. However, in polynomial structure, multiplication becomes a nontrivial task. Three different algorithms for multiplication are investigated based upon the basis representation of polynomial form in finite fields. Finally, the concept of discrete Fourier transforms in finite field are reviewed to be applied to the development of the fast Galois field transforms in the following chapters.

3.2 Index Calculus

Index of β Relative to α ($\mathrm{ind}_\alpha\beta$ or $\log_\alpha\beta$). Let α be a primitive root of N and $\gcd(\beta, N) = 1$, then k is the index of β relative to α (written $k = \mathrm{ind}_\alpha\beta$) if it is the smallest positive integer such that $\beta \equiv \alpha^k \bmod N$. The utility of indices is due to the following logarithm-like relationships they obey:

$$\mathrm{ind}_\alpha ab \equiv (\mathrm{ind}_\alpha a + \mathrm{ind}_\alpha b) \bmod \phi(N) ;$$

$$\mathrm{ind}_\alpha b^l \equiv (l\ \mathrm{ind}_\alpha b) \bmod \phi(N) ;$$

$$\mathrm{ind}_\alpha 1 \equiv 0 \bmod \phi(N) ;$$

$$\mathrm{ind}_\alpha \alpha \equiv 1 \bmod \phi(N) .$$

For example, Table 3.1 shows the indices of numbers relatively prime to $N = 9$. According to

Table 3.1 Illustration of Indices for $N = 9$

$k = \mathrm{ind}_2 a$	1	2	3	4	5	6
$a = 2^k \bmod 9$	2	4	8	7	5	1

the table and the fact that $\phi(9) = 9(1 - 1/3) = 6$, the following cases obtained

1) $\mathrm{ind}_2 8 \cdot 7 = \mathrm{ind}_2 8 + \mathrm{ind}_2 7 \equiv 1 \bmod 6$, and $2^{\mathrm{ind}_2 8 \cdot 7} = 2^1 \equiv 8 \cdot 7 \bmod 9$;

2) $\mathrm{ind}_2 8 = \mathrm{ind}_2 2^3 = 3\,(\mathrm{ind}_2 2) = 3$;

3) $\mathrm{ind}_2 1 = 6 \equiv 0 \bmod 6$; and

4) $\mathrm{ind}_2 2 = 1$. ♦

3.3 The Table Lookup Method For Small Fields

In the case of small finite fields, it is more efficient to use the power form to represent field elements while applying the table lookup method for its arithmetic operations. Fast memory devices are good candidates for the implementation of tables as long as the table capacity does not exceed the current device technology. The following algorithms are used to perform the arithmetic operations in finite fields.

All-Table (AT) approach. Tables can be implemented using fast memory devices, such as programmable read-only memory (PROM) and static random-access memory (SRAM), with $r_A + r_B$ bits of address line and $r_{A \cdot B}$ bits of data output line. The notations r_A and r_B represent the number of bits of two input operands A and B respectively, and $r_{A \cdot B}$ represents the number of bits of the output under an operation of A and B. For $i = A$, B, or AB, $r_i = \log_2 q_i$ where q_i is the range of data in the finite field GF(q).

In this approach, the memory capacity requirement which is regarded as the number of gates of the memory devices is calculated approximately as

$$\begin{aligned} T &= 2^{(r_A + r_B)} \cdot r_{A \cdot B} \\ &= q_A \cdot q_B \cdot \log_2 q_{A \cdot B} \sim O(q^2 \log_2 q) . \end{aligned}$$

An implementation of a finite field arithmetic unit is depicted in Figure 3.1 where the precomputed contents based on the type of the arithmetic operation is loaded to the lookup table memory initially.

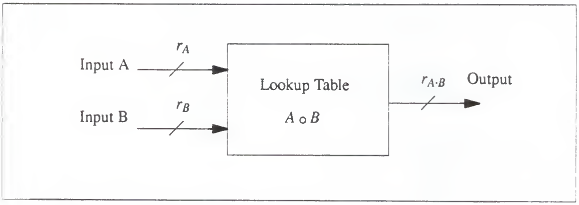


Figure 3.1 A Finite Field Arithmetic Unit Based On the All-Table Approach

Mixed Table-Logic (MTL) approach. The capacity requirement on the factor of q^2 in the AT approach prevent the operations from full application of the table lookup method. Let α be a primitive element and β be a nonzero element of GF(q), the index of β with respect to the base α is the uniquely determined integer r where $0 \le r \le q-1$ and $\beta = \alpha^r$ from the construction of finite fields discussed in previous section,. The index r is called the discrete logarithm of β and denoted by $r = \log_\alpha \beta$ (or $\mathrm{ind}_\alpha \beta$). Its inverse function, denoted as $\exp_\alpha(r) = \alpha^r = \beta$, is called the discrete exponentiation. Applying the isomorphism

$$\log_\alpha \ : \ \mathrm{GF}(\ q\ )^* \ \rightarrow \ \mathbf{Z}_{q-2}, \tag{3.1}$$

for a primitive element $\alpha \in \mathrm{GF}(\ q\ )$, multiplication can be transformed into addition modulo $q-1$ — provided that logarithm and exponentiation tables are available.

A multiplier based on the discrete logarithm/exponentiation approach is shown in Figure 3.2 . Note that the number of gates required for a suitable memory is given by $O(q\log_2 q)$.

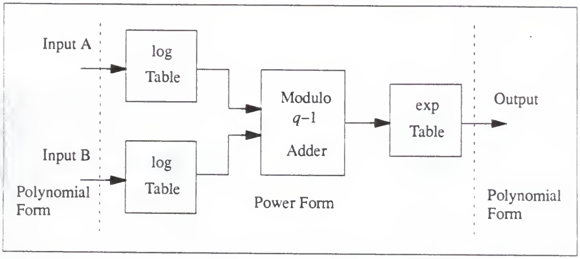


Figure 3.2 A Discrete Logarithm/Exponentiation Multiplier

Comparing to the AT approach, a q-fold saving in terms of memory size is obtained which becomes more significant when the order q of a field becomes large.

When the elements of a field are represented in logarithmic format, addition becomes a nontrivial task. Applying group property of GF(q)*, Zech's logarithms [Con68] suggests an efficient way to perform exponential addition by introducing a second level of table lookup. Let $M_q = \{\ 0, 1, \ldots, q-2\ \} \cup \{\ -\infty\ \}$ in GF(q), Zech's logarithm, denoted as $Z(n)$, is defined as

$$Z(n) = \log_\alpha(1 + \alpha^n) \tag{3.2}$$

for all elements $n \in \mathcal{M}_q$. It is obvious that Z is a mapping $Z : \mathcal{M}_q \rightarrow \mathcal{M}_q$. Therefore, addition performs as follows: Let z be the exponent of the sum, $\alpha^x + \alpha^y$, which is $\alpha^z = \alpha^x + \alpha^y$, then

$$\alpha^z = \alpha^x (1 + \alpha^{y-x})$$

$$= \alpha^y (1 + \alpha^{x-y}) \,. \tag{3.3}$$

Applying Zech's logarithm defined in Equation (3.2), the exponent of the sum becomes

$$z = x + Z(y-x) = y + Z(x-y). \tag{3.4}$$

Example 3.1 The Zech's logarithms in GF(2^4) and their addition operation.

Zech's logarithms in the finite field GF(2^4) which is generated by the polynomial $f_1(x)$ as is in Example 2.1 are listed in Table 3.2.

Table 3.2 The Zech's Logarithms in GF(2^4)

n	$-\infty$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Z(n)	0	$-\infty$	4	8	14	1	10	13	9	2	7	5	12	11	6	3

Accordingly, the example of addition of elements α^3 and α^7 is given as

$$\alpha^3 + \alpha^7 = \alpha^3 (1 + \alpha^4) = \alpha^3 \alpha^{Z(4)} = \alpha^4 \,. \ \blacklozenge$$

Having the nontrivial addition problem solved, the arithmetic operations in a small finite field GF(q) can be carried out rather easily. Let inputs $A = \alpha^x$ and $B = \alpha^y$, and the isomorphism in Equation (3.1) hold, then the arithmetic operations of the finite field GF(q) are summarized as follows.

1) Addition:

$$A + B \mapsto [x + Z(y-x)] \bmod (q-1) \,, \tag{3.5}$$

or

$$A + B \mapsto [y + Z(x-y)] \bmod (q-1) \,; \tag{3.6}$$

2) Additive Inverse: Since $p-1 \equiv -1$ in GF(q) of characteristic p, discrete logarithm function is used to find k such that $\alpha^k = p-1$. Then, the inverse of B becomes

$$-B = -\alpha^y = \alpha^k \cdot \alpha^y = \alpha^{(k+y) \bmod q-1} . \tag{3.7}$$

Therefore, the isomorphic mapping is

$$-B \mapsto [k+y] \bmod (q-1) ; \tag{3.8}$$

3) Subtraction: According to the fact that $\alpha^k = p-1$,

$$A - B \mapsto [x + Z(k+y-x)] \bmod (q-1) ; \tag{3.9}$$

4) Multiplication:

$$A \cdot B \mapsto [x+y] \bmod (q-1) ; \tag{3.10}$$

5) Multiplicative Inverse: Since $B^{-1} = \alpha^{-y} = \alpha^{q-1-y}$, thus

$$B^{-1} \mapsto q - y - 1 ; \tag{3.11}$$

6) Division:

$$A \mathbin{/} B \mapsto [x + (q-1-y)] \bmod (q-1) ; \tag{3.12}$$

7) Power:

$$A^n \mapsto nx \bmod (q-1) . \tag{3.13}$$

Logarithmic MTL arithmetic unit. A finite field arithmetic unit based on power form representation of field element is developed and shown in Figure 3.3. The logarithm tables in the front end serve as the isomorphic mapping defined in Equation (3.1) whereas the exponentiation table in the back end serves as the isomorphic inverse mapping. In the discrete logarithm domain of the arithmetic unit, operations such as addition, subtraction, multiplication, and division are carried out based on the algorithms developed in the preceding section. The type of arithmetic operation is chosen by controlling the 'mode selection' signal at the multiplexer of the output stage. Since the bypass of the Zech's logarithm table lookup which

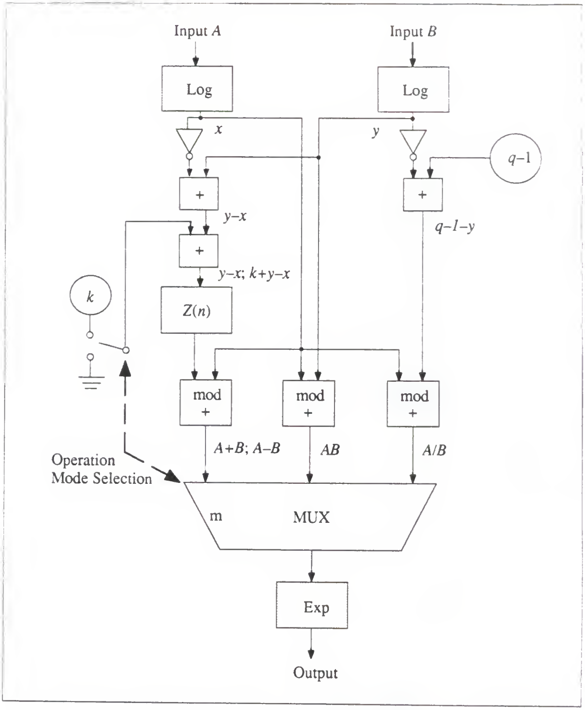


Figure 3.3 A Logarithmic Finite Field Arithmetic Unit

related to those additive operations, the multiplication operation is the fastest among all other ones.

Computation of Zech's logarithm table Since α is a primitive element in GF(q), x and α^x are in one-to-one correspondence for $x = -\infty, 0, 1, \ldots, q-2$. The relationship, accordingly, also exists between x and $Z(x)$, hence Zech's logarithm is a bijective mapping. By knowing x, Zech's logarithm can be obtained from the brute-force approach according to their definition in Equation (3.2).

Provided that the logarithm or exponentiation tables as those shown in Table 3.1 or Table 3.2 are available, the following algorithm is used to compute the Zech's logarithm table.

Algorithm 3.1

1) The vector form, which is $[\ x_0,\ x_1,\ \ldots,\ x_{n-1}\ ]$, of the field element α^x is obtained from the precomputed exponentiation table;

2) Using modulo p arithmetic where p is the characteristic of GF(q), add 1 to the least significant tuple of the vector form of α^x. This leads to the fact that the least significant tuple becomes $x_0' \equiv x_0 + 1 \mod p$;

3) Zech's logarithm $Z(x)$ is then obtained by applying the the resultant vector $[\ x_0',\ x_1',\ \ldots,\ x_{n-1}'\ ]$ to the logarithm table. ♦

Although these logarithms are useful for computation purposes, the construction of the table of $Z(x)$ is rather cumbersome. However, by applying the concept of the cyclotomic cosets along with the properties of $Z(x)$ introduced by Imamura [Ima80] and Huber [Hub90], a time-saving method can then be developed.

Some properties of $Z(x)$ are first reviewed: If $\alpha^x \neq 0$, that is $x \neq -\infty$, then $\alpha^{Z(q-1-x)} = 1 + \alpha^{q-1-x}$. Since $\alpha^{q-1} = 1$, it turns out that $\alpha^{Z(q-1-x)} = 1 + \alpha^{-x} = (\ \alpha^x + 1\ )\ \alpha^{-x} = \alpha^{Z(x)-x}$. Therefore,

$$Z(\ q-1-x\ ) \equiv [\ Z(x) - x\ ] \bmod (\ q-1\ ). \tag{3.14}$$

From Theorem 2.3, the relationship $(1 + \alpha^x)^p = 1 + \alpha^{px}$ holds in GF(q) which leads to the result that $\alpha^{Z(px)} = 1 + \alpha^{px} = (1 + \alpha^x)^p = \alpha^{pZ(x)}$. Therefore,

$$Z(px) \equiv pZ(x) \bmod (q-1). \tag{3.15}$$

The inverse function $Z^{-1}(x)$ of Zech's logarithms is derived by defining the following iterated function $Z^{-i} : M_q \rightarrow M_q$ and

$$Z^i(x) = Z(Z^{i-1}(x)) \tag{3.16}$$

where $i = 2, 3, \ldots$ and $Z^1 = Z(x)$. Obviously, $\alpha^{Z^i(x)} = i + \alpha^x$ and $\alpha^{Z^p(x)} = \alpha^x$. Thus, $Z(Z^{-1}(x))$ $= x = Z^p(x) = Z(Z^{p-1}(x))$. Accordingly, the inverse function $Z^{-1}(x)$ becomes

$$Z^{-1}(x) = Z^{p-1}(x) . \tag{3.17}$$

When $p = 2$, $Z^{-1}(x) = Z(x)$. However, for $p \neq 2$, $Z^{-1}(x)$ is derived as follows. By choosing r such that $\alpha^r = 2i$ and from Equation (3.16), $\alpha^{Z^i(x)} = i + \alpha^x = 2i + p - i + \alpha^x = \alpha^r (1 + \alpha^{Z^{p-i}(x)-r}) = \alpha^r \cdot \alpha^{Z(Z^{p-i}(x)-r)}$. Thus,

$$Z^i(x) = r + Z(Z^{p-i}(x) - r). \tag{3.18}$$

For the case $i = p-1$, $\alpha^r = p-2$, Equation (3.18) becomes

$$Z^{p-1}(x) = Z^{-1}(x) = r + Z (Z(x) - r). \tag{3.19}$$

The cyclotomic coset concept is linked to the calculation of $Z(x)$ by the property stated in Equation (3.15). For the coset $C_s = \{ s, sp, sp^2, \ldots, sp^{l_s-1} \}$, and the known logarithm $Z(s)$, then the corresponding mapped coset is

$$C_s' = \{ Z(s), pZ(s), p^2Z(s), \ldots, p^{l_s-1}Z(s) \}.$$

Note that Zech's logarithm maps cosets onto cosets of the same length. Having the known logarithm $Z(s)$, by either repeatedly applying Equation (3.14) or the inverse function in Equation (3.17) or (3.19), all other cosets of the same length are obtained. Thus, by simply computing the key element $Z(s)$ for the set of cosets of the same length, the Zech's logarithm table will be constructed easily.

Example 3.2 Computation of the table of $Z(x)$.

Consider the field GF(p^m) = GF(2^4), constructed with the primitive polynomial $f_1(x) = x^4 + x + 1$. From $f_1(x)$ the key element $Z(1) = 4$ is obtained immediately since $x^{Z(1)} = 1 + x^1 \equiv x^4 \bmod f_1(x)$. Using Equations (3.15) and (3.17) gives the Zech's logarithm of all elements in C_1, where $C_1 = \{ 1, 2, 4, 8 \}$. $Z(1) = 4, Z(4) = 1, Z(2) = 8, Z(8) = 2$. Having $Z(1) = 4$, applying Equation (3.14), we find $Z(14) = 3$ also $Z(3) = 14$. Hence elements in $C_3 = \{ 3, 6, 12, 9 \}$ will be one-to-one corresponding to the set $\{ 14, 13, 11, 7 \}$. Finally, for the only coset of length 2, $C_5 = \{ 5, 10 \}$, we immediately know $Z(5) = 10$, $Z(10) = 5$. The result meets Table 3.2.

Another example, for field GF(2^5) with defining polynomial $f(x) = x^5 + x^2 + 1$, we get the key element $Z(2) = 5$. The following list represent the procedure to find the Zech's logarithm table.

1) $C_1 = \{ 1, 2, 4, 8, 16 \}$;

2) Apply (3.15) to $Z(2) = 5$, $C_5 = \{ 18, 5, 10, 20, 9 \}$ is obtained;

3) Apply (3.14) to $Z(18) = 1$, $Z(13) = -17 = 14$ is obtained;

4) Apply (3.15) to $Z(13) = 14$, $C_{11} = \{ 13, 26, 21, 11, 22 \}$ is obtained;

5) $C_7 = \{14, 28, 25, 19, 7 \}$;

6) Apply (3.14) to $Z(14) = 13$, $Z(17) = -1 = 30$ is obtained;

7) $C_3 = \{ 17, 3, 6, 12, 24 \}$;

8) Apply (3.15) to $Z(17) = 30$, $C_{15} = \{ 30, 29, 27, 23, 15 \}$ is obtained. ♦

The algorithm based upon the concept of cyclotomic coset has improved the calculation of Zech's logarithm. The brute force method takes p^m calculation for the finite field GF(p^m) while the new algorithm takes k operation where k is the number of cosets of the different coset length.

3.4 Discrete Logarithm and Exponentiation Problems

If a finite field is small enough, one can tabulate all the field elements and their logarithms, and use this table for computation within the field, much as one uses a table of natural logarithms for calculations involving real numbers. However, for large fields, for instance that of $\mathrm{GF}(2^{127})$, it is infeasible to tabulate its logarithms. Thus the construction of discrete logarithmic and exponential tables presents a great challenge. Furthermore, from previous discussions, the table lookup method requires a memory capacity on the order of $p^m \log_2 p^m$. As for the current state of the art on semiconductor device technology, high speed megabit memories can only provide capacity to the fields of order $p^m = 10^6$.

Aside from the intrinsic interest that the problem of computing discrete logarithms has, it is of considerable importance in cryptography [Adl79, Blak84]. The use for cryptography depends on the apparent one-way nature of the discrete logarithm function: it is relatively difficult to extract logarithms in a large field, which it is relatively easy to exponentiate. Several proposed algorithms for computing discrete logarithms are known. For the example that the field $\mathrm{GF}(2^{127})$ and the primitive polynomial involved is $f(x) = x^{127} + x + 1$, Adleman's algorithm [Adl79] seems to take two weeks; a modification due to Blake [Blak84] takes about nine hours.

The discrete exponential function $\exp_\alpha(r) = \alpha^r$ in F_q (or $\mathrm{GF}(q)$)can be calculated for $1 \le r \le q-1$ by the algorithm which is often called the square-and-multiply technique. In detail, the elements α , $\alpha^2, \alpha^4, \dots, \alpha^{2^e}$ are first computed by repeated squaring, where 2^e is the largest power of 2 that is smaller than r. Then α^r is obtained by multiplying together an appropriate combination of these elements. For instance, to get α^{27} one would multiply together the elements α , α^2, α^8 and α^{16}. A simple analysis shows that the calculation of α^r requires at most $2\lfloor \log_2 q \rfloor$ multiplications in F_q . Until recently, the inverse problem of computing discrete logarithms in F_q was believed to be much harder, since for one of the best algorithms available then the required number of arithmetic operations in F_q was of the order of magnitude $q^{\frac{1}{2}}$. If the order q is sufficiently large, for instance $q > 2^{100}$, exponentiation in F_q might justly have been regarded as a one-way function. However, great progress has recently been achieved [Blak84, Lid86] in the computation of discrete logarithms, which makes it necessary to construct cryptosystems based on discrete exponentiation in a careful manner in order to prevent them against attacks by these recent algorithms.

A discrete logarithm algorithm for F_q is now discussed, which is the index-calculus algorithm [Blak84]. Suppose α is a primitive element of F_q where $q = p^m$. The algorithm to find the discrete logarithm $\log_\alpha \beta$ of an arbitrary nonzero element β of F_q consists of two stages. The first stage which generates precomputed results serves as a data base for the second stage of the algorithm.

Algorithm 3.2 The Computation of the Discrete Logarithm of F_{p^m}.

1) Precomputation Stage:

1.1) Compute the discrete logarithms of all elements of a chosen subset V of F_q. The set V usually consists of all the monic irreducible polynomials over F_p of degree $\leq s$, where $s < m$.

1.2) $\log_\alpha d$ is known for all $d \in F_p^*$. For $p = 2$, $\log_\alpha(1) = 0$. For $p > 2$ we use the observation that $\alpha' = \alpha^{(q-1)/(p-1)}$ is a primitive element of F_p (because $\alpha'^{(p-1)} = \alpha^{q-1} = 1$), and $d = (\alpha')^{\log_{\alpha'} d} = \alpha^{[(q-1)/(p-1)] \log_{\alpha'} d}$. Such that

$$\log_\alpha d = [(q-1)/(p-1)] \log_{\alpha'} d \text{ , for all } d \in F_p^* .$$

The value $\log_{\alpha'} d$ may be obtained by direct calculation.

1.3) Choose a random integer, t, with $1 \leq t \leq q-2$, and form

$$\alpha_1 = \alpha^t \bmod f(x), \ \deg(\alpha_1) < m.$$

Then factor α_1 into irreducible polynomials over F_p . If all the monic irreducible factors are elements of V, so that

$$\alpha_1 = d \prod_{v \in V} v^{e_v(\alpha_1)} \tag{3.20}$$

with $d \in F_p^*$ is the canonical factorization in $F_p\,[x]$, then

$$t = [\ \log_\alpha(d) + \sum_{v \in V} e_v(\alpha_1) \log_\alpha v\ ] \bmod (q-1)\ . \tag{3.21}$$

2) Main Stage:

2.1) Similar to 1.3), form the polynomial $\beta_1 \in F_p\,[x]$, $\beta_1 \equiv \beta\,.\alpha^t \bmod f(x)$, $\deg(\beta_1) < m$, and thus

$$\log_\alpha \beta = [\ \log_\alpha(d) + \sum_{v \in V} e_v(\beta_1) \log_\alpha v - t\ ] \bmod (q-1)\,.$$

2.2) If β_1 does not have the desired type of factorization, choose other values of t until this type of factorization is obtained. ♦

Example 3.3 Compute the Discrete logarithm of $\beta = x^4 + x^3 + x^2 + x + 1$ in F_{64}.

We consider F_{64} defined by $f(x) = x^6 + x + 1 \in F_2\,[x]$. Then take $\alpha = x$ as a primitive element of F_{64} since $f(x)$ is a primitive polynomial over F_2 . First we compute the data base of V.

1) Suppose the maximum degree s of irreducible polynomials in the set V is 2 such that $V = \{ x, x+1, x^2 + x + 1 \}$. Apparently $\log_x(x) = 1$. We choose integer t with $1 \le t \le 62$. A good choice is $t = 6$, since $\alpha_1 = \alpha^t = x^6 \equiv (x+1) \bmod f(x)$, hence $\log_\alpha(x+1) = 6$. Another good choice is $t = 32$, since $x^{64} \equiv x \equiv x^6 + 1 \equiv (x^3 + 1)^2 \bmod f(x)$ implies

$$\alpha_1 \equiv (x^3 + 1) \equiv (x+1)(x^2 + x + 1) \bmod f(x)\ .$$

This yields

$$32 \equiv \log_\alpha(x+1) + \log_\alpha(x^2 + x + 1) \equiv 6 + \log_\alpha(x^2 + x + 1) \bmod 63\ ,$$

hence $\log_x(x^2 + x + 1) = 26$.

2) The choice of $t = 2$ yields

$$\beta_1 \equiv (x^4 + x^3 + x^2 + x + 1)x^2 \equiv (x^6 + x^5 + x^4 + x^3 + x^2)$$

$$\equiv x^5 + x^4 + x^3 + x^2 + x + 1 \equiv (x^2 + x + 1)^2(X + 1) \bmod f(x) ,$$

hence all the irreducible factors are in V. Therefore

$$\log_x(\beta) \equiv 2 \log_x(x^2 + x + x + 1) + \log_x(x + 1) - 2$$

$$\equiv 2 \cdot 26 + 6 - 2 \equiv 56 \bmod 63 ,$$

and so $\log_x(x^4 + x^3 + x^2 + x + 1) = 56$. ♦

Up to this point, the computations in finite field, especially multiplication, are easily performed by using power form representation. However, the problems of discrete logarithm and exponentiation offset the advantages when the order of a field becomes large. Therefore, the computations of finite fields in vector forms are considered. Under such formats, operations like addition and subtraction are trivial since they compute in a component-wise fashion over the ground field GF(p). Whereas, multiplication and division are more complicated and time-consuming. Based upon the representation of field elements in vector form, there are various methods in designing multiplier in finite fields. In next section, the architectures based on primal, dual, and normal bases are investigated. These designs can be applied and integrated to the finite field computation systems which will be developed in later chapters.

In the finite field GF(p^m), the most common type of circuit uses a linear feedback shift register to form the desired product sequentially in m clock cycles. This circuit is simple and relatively economical, but it operates rather slowly because of the m flip-flop delays incurred. The possible solutions to the slowness due to the sequential operation are also given. A performance analysis for each design architectures is investigated to show the design trade-off of these architectures. The representation of the components in circuit diagram of finite field multiplier is shown in Figure 3.4.

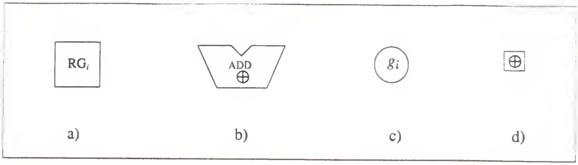


Figure 3.4 The Building Block of the Finite Field Multipliers. a) Storage element such as register and accumulator. b) Multiple input Modulo p adder. c) Modulo p multiplier for multiplying a data with g_i. d) Modulo p adder.

3.5 Primal Basis Multiplier

Let $a \in \mathrm{GF}(p^m)$ be expressed in primal coordinates as

$$a = a_0 + a_1\alpha + a_2\alpha^2 + \ldots + a_{m-1}\alpha^{m-1} \tag{3.22}$$

where α is a primitive element in GF(p^m) and N_p be a primal basis generated by α . Assume the irreducible defining polynomial is

$$f(x) = f_0 + f_1x + f_2x^2 + \ldots + f_{m-1}x^{m-1} + x^m \, . \tag{3.23}$$

Such that $f(\alpha\ ) = 0$ or

$$\alpha^n = (-f_0) + (-f_1)\alpha + (-f_2)\alpha^2 + \ldots + (-f_{m-1})\alpha^{m-1} \tag{3.24}$$

is used to reduce a polynomial to the one with degree less than m. Before the calculation of the product of elements in GF(p^m), the multiplication of a field element a and the primitive element α and its power α^l is derived first. Since the product of a and α is

$$a\alpha = a_0\alpha + a_1\alpha^2 + a_2\alpha^3 + \ldots + a_{m-2}\alpha^{m-1} + a_{m-1}\alpha^m \, . \tag{3.25}$$

By applying Equation (3.24) and let $g_i = -f_i$, the product $a\alpha$ becomes

$$\begin{aligned} a\alpha &= (-f_0a_{m-1}) + (a_0 - f_1a_{m-1})\alpha + (a_1 - f_2a_{m-1})\alpha^2 + \ldots + (a_{m-2} - f_{m-1}a_{m-1})\alpha^{m-1} \\ &= (g_0a_{m-1}) + (a_0 + g_1a_{m-1})\alpha + (a_1 + g_2a_{m-1})\alpha^2 + \ldots + (a_{m-2} + g_{m-1}a_{m-1})\alpha^{m-1} \, . \end{aligned} \tag{3.26}$$

This relationship suggests an implementation of the product $a\alpha$ by an m-digit shift register array, RG$_i$'s, and a feedback network which routes the most significant digit a_{m-1} to the rest of the digits according to the polynomial reduction mechanism. A generic primal basis $a\alpha$ - multiplier is shown in Figure 3.5. In terms of propagation delay and hardware cost, this im-

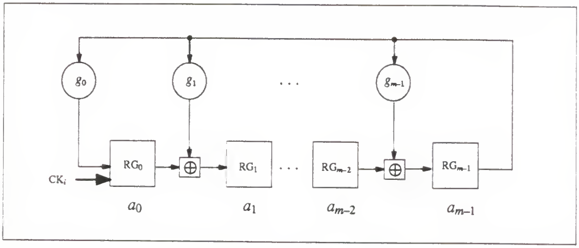


Figure 3.5 A Generic Primal Basis $a\alpha$ - Multiplier

plementation results in a one level time delay of modulo addition and j modulo p multipliers and adders, where j is the number of nonzero g_i.

In the case of $1< l < p^m -1$, the evaluation of $a\alpha^l$ is achieved simply by clocking the registers continuously to have l times right shifts, then the final result will stay in the registers. Thus, the product $a\alpha^l$ can be obtained at most in $m-1$ clocks. This is achieved by the following mechanism.

From the fact that the defining polynomial $f(x)$ is used to reduce a polynomial to the degree less than m, α^l can always be represented in a primal basis. Thus $a\alpha^l$ has the polynomial form as

$$\begin{aligned} a\alpha^l &= a\ (\ h_0 + h_1\alpha + \ldots + h_{m-2}\alpha^{m-2} + h_{m-1}\alpha^{m-1}\ ) \\ &= ah_0 + ah_1\alpha + \ldots + ah_{m-2}\alpha^{m-2} + ah_{m-1}\alpha^{m-1} \\ &= h_0 \cdot a + h_1 \cdot a\alpha + \ldots + h_{m-2} \cdot a\alpha^{m-2} + h_{m-1} \cdot a\alpha^{m-1}\ . \end{aligned} \tag{3.27}$$

This equation can be interpreted as that $a\alpha^l$ is obtained by summing the cyclic shifted terms $h_i \cdot a\alpha^i$ where $i = 0, 1, \cdots, m-1$. Therefore, $a\alpha^l$ is obtained in $m-1$ clocks even l may be the value up to $p^m - 2$. The first algorithm for implementing the multiplication is listed below:

Algorithm 3.3 Primal Basis Accumulated Multiplication.

```
INPUT:   [ $a_0, a_1, \ldots a_{m-2}, a_{m-1}$ ] ( Parallel–In )
         [ $h_0, h_1, \ldots h_{m-2}, h_{m-1}$ ] ( Serial–In; $h_i$ first in )
0. Initialization:
                 $RG_i \leftarrow a_i$, for $i = 0, 1, \cdots, m-1$.
                 $AC_i \leftarrow 0$;
1. Clock 1:      $RG_i \leftarrow a\alpha \bmod f(\alpha)$,
                 $AC_i \leftarrow h_0 \cdot a$;
2. Clock 2:      $RG_i \leftarrow a\alpha^2 \bmod f(\alpha)$,
                 $AC_i \leftarrow [h_1 \cdot a\alpha] \bmod f(\alpha)$,
                 $AC_i = [h_0 \cdot a + h_1 \cdot a\alpha] \bmod f(\alpha)$,
                   ⋮
j. Clock j:      $RG_i \leftarrow a\alpha^j \bmod f(\alpha)$,
                 $AC_i \leftarrow [h_{j-1} \cdot a\alpha^{j-1}] \bmod f(\alpha)$,
      $AC_i = [h_0 \cdot a + h_1 \cdot a\alpha + h_2 \cdot a\alpha^2 + \ldots + h_{j-1} \cdot a\alpha^{j-1}] \bmod f(\alpha)$,
                   ⋮
m–1. Clock m–1:
                 $RG_i \leftarrow a\alpha^{m-1} \bmod f(\alpha)$,
                 $AC_i \leftarrow [h_{m-2} \cdot a\alpha^{m-2}] \bmod f(\alpha)$,
```

$$AC_i = [\ h_0 \cdot a + h_1 \cdot a\alpha + h_2 \cdot a\alpha^2 + \ldots + h_{m-2} \cdot a\alpha^{m-2}\ ] \bmod f(\alpha\ );$$

m. Clock *m*: $RG_i \leftarrow a\alpha^m \bmod f(\alpha\ ),$

$AC_i \leftarrow [\ h_{m-1} \cdot a\alpha^{m-1}\ ] \bmod f(\alpha\ ),$

$$AC_i = [\ h_0 \cdot a + h_1 \cdot a\alpha + h_2 \cdot a\alpha^2 + \ldots + h_{m-1} \cdot a\alpha^{m-1}\ ] \bmod f(\alpha\ ).$$

OUTPUT: Stored in the accumulator ACs (Parallel–Out). ♦

Thus, a generic primal basis accumulated multiplier based on the algorithm is developed and shown in Figure 3.6. The components h_i of α^l are sequentially clocked to be avail-

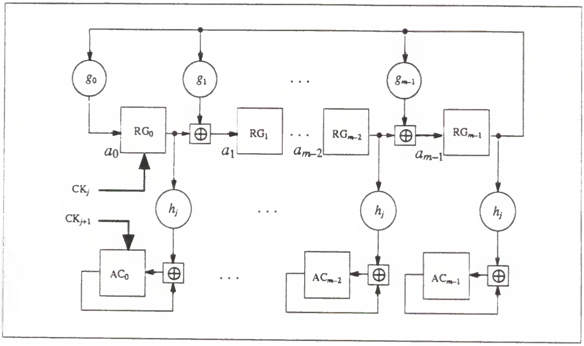


Figure 3.6 A Primal Basis Accumulated $a\alpha^l$-Multiplier

able for processing and a is simultaneously loaded to the register at the initial clock. The contents of the register is updated by the most significant digit (MSD) of the feedback digit at clock i and the component of $a\alpha^l$ are accumulated at clock i+1. Iterating such procedures, all the component of $a\alpha^l$ will simultaneously be available at mth clock. The summing operation is done locally at each component of $a\alpha^l$ by the accumulator ,ACs, therefore, k copies of the accumulators where k is the number of nonzero of h_i exist. The architecture of this design is regular and the advantage of this design is high speed processing capability in the output stage due to the single level delay. To compute the product of a and b where $a, b \in \mathrm{GF}(p^m)$, let a is represented as in Equation (3.22), while b is represented similarly as

$$b = b_0 + b_1\alpha + \ldots + b_{m-2}\alpha^{m-2} + b_{m-1}\alpha^{m-1},$$

then

$$ab = ab_0 + ab_1\alpha + \ldots + ab_{m-2}\alpha^{m-2} + ab_{m-1}\alpha^{m-1}$$

$$= b_0 \cdot a + b_1 \cdot a\alpha + \ldots + b_{m-2} \cdot a\alpha^{m-2} + b_{m-1} \cdot a\alpha^{m-1}.$$

This is exactly the form of Equation (3.27). Thus, a ab-multiplier is realized by applying the same design as that in Figure 3.6. Another implementation of ab-multiplier is based on nested form of polynomial multiplication and is demonstrated in the following algorithm:

Algorithm 3.4 Primal Basis Nested Multiplication.

```
INPUT:   [ $a_0, a_1, \ldots a_{m-2}, a_{m-1}$ ] ( Parallel–In )
         [ $b_0, b_1, \ldots b_{m-2}, b_{m-1}$ ] ( Serial–In; $b_{m-1}$ first in )
0. Initialization:
         $RG_i \leftarrow 0$, for $i = 0, 1, \ldots, m-1$;
1. Clock 1:   $RG_i \leftarrow b_{m-1} \cdot a$;
2. Clock 2:   $RG_i \leftarrow [\,(b_{m-1} \cdot a) \cdot \alpha + b_{m-2} \cdot a\,] \bmod f(\alpha)$;
3. Clock 3:
         $RG_i \leftarrow [\,(b_{m-1} \cdot a \cdot \alpha + b_{m-2} \cdot a) \cdot \alpha + b_{m-3} \cdot a\,] \bmod f(\alpha)$,
              ⋮
j. Clock j:
   $RG_i \leftarrow [\,(b_{m-1} \cdot a \cdot \alpha^{j-2} + \ldots + b_{m-j+1} \cdot a) \cdot \alpha + b_{m-j} \cdot a\,] \bmod f(\alpha)$,
              ⋮
```

m. Clock *m*:

$$\begin{aligned} \mathrm{RG}_i \leftarrow & [(b_{m-1} \cdot a \cdot \alpha^{m-2} + \ldots + b_1 \cdot a) \cdot \alpha + b_0 \cdot a] \bmod f(\alpha), \\ = & [b_{m-1} \cdot a \cdot \alpha^{m-1} + \ldots + b_1 \cdot a \cdot \alpha + b_0 \cdot a] \bmod f(\alpha), \\ = & [a \cdot (b_{m-1} \cdot \alpha^{m-1} + \ldots + b_1 \cdot \alpha + b_0)] \bmod f(\alpha), \\ = & [a \cdot b] \bmod f(\alpha). \end{aligned}$$

OUTPUT: Stored in the register RGs (Parallel–Out). ♦

A generic primal basis *ab*-multiplier is developed and shown in Figure 3.7. There is only one

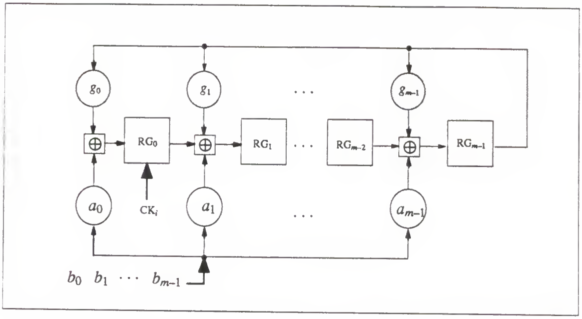


Figure 3.7 A Primal Basis Nested *ab* Multiplier

set of registers needed to store the intermediate nested product. One of the input *b* is fed into the system sequentially by starting with the MSD, b_{m-1} . Iterating such procedures, all the component of *ab* will simultaneously be available at *n*th clock. The architecture of this design is simple and less hardware. However the processing clock cycle is of longer due to the double level delay in the three input modulo *p* adders.

3.6 Dual Basis Multiplier

For N_p a primal basis and α a primitive element in GF(p^m), there exists a dual basis with respect to N_p. Let a be expressed in the dual coordinates as

$$a = a_0'\beta_0 + a_1'\beta_1 + \ldots + a_{m-1}'\beta_{m-1}. \tag{3.28}$$

Since the element α_j of N_p is α^j in polynomial basis, we know from Equation (3.25) that

$$a_j' = \mathrm{Tr}(a\alpha_j) = \mathrm{Tr}(a\alpha^j) . \tag{3.29}$$

To evaluate $a\alpha^l$ in the dual coordinate system, we start from the case of $l = 1$.

$$a\alpha = (a\alpha)_0'\beta_0 + (a\alpha)_1'\beta_1 + \ldots + (a\alpha)_{m-1}'\beta_{m-1} .$$

For $j = 0, 1, \ldots, m-2$, from Equation (3.29), one have

$$(a\alpha)_j' = \mathrm{Tr}(a\alpha \cdot \alpha_j) = \mathrm{Tr}(a\alpha^{j+1}) = a_{j+1}' . \tag{3.30}$$

Furthermore for $j = m-1$

$$(a\alpha)_{m-1}' = \mathrm{Tr}(a\alpha \cdot \alpha^{m-1}) = \mathrm{Tr}(a\alpha^m) . \tag{3.31}$$

By applying Equation (3.29) and the polynomial reduction as that shown in Equations (3.24) and (3.26), Equation (3.31) becomes

$$\begin{aligned}(a\alpha)_{m-1}' &= \mathrm{Tr}(\ a \cdot [(-f_0) + (-f_1)\alpha + (-f_2)\alpha^2 + \ldots + (-f_{m-1})\alpha^{m-1}]\ ) \\ &= \mathrm{Tr}(-f_0a) + \mathrm{Tr}(-f_1a\alpha) + \mathrm{Tr}(-f_2a\alpha^2) + \ldots + \mathrm{Tr}(-f_{m-1}a\alpha^{m-1}) \\ &= (-f_0)\mathrm{Tr}(a) + (-f_1)\mathrm{Tr}(a\alpha) + (-f_2)\mathrm{Tr}(a\alpha^2) + \ldots + (-f_{m-1})\mathrm{Tr}(a\alpha^{m-1}) \\ &= g_0a_0' + g_1a_1' + g_2a_2' + \ldots + g_{m-1}a_{m-1}' .\end{aligned} \tag{3.32}$$

The relationships in Equations (3.30) and (3.32) suggest a simple implementation of the multiplication by a shift register array and a feedback network which is a tree of modulo p adders. It is obvious from above equations that the burden of the operation, both in time and cost, is heavily located on the evaluation of the most significant digit $(a\alpha)_{n-1}'$. In terms of propagation delay and hardware cost, this implementation results a d level delay of addition time with

$d = \lceil \log k \rceil$ and k modulo p adders required where k is the number of nonzero g_i. Figure 3.8

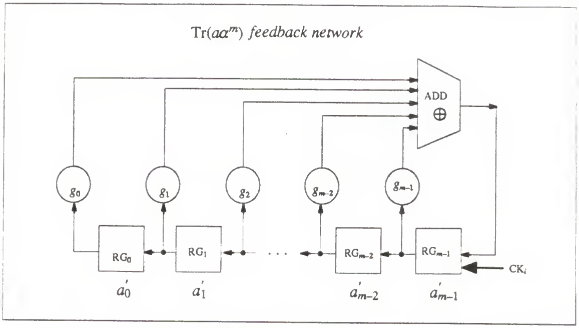


Figure 3.8 A Generic Dual Basis $a\alpha$ -Multiplier

shows a generic dual basis $a\alpha$ -multiplier where the circuitry of the feedback network Tr($a\alpha^m$) is defined by $f(x)$ as in Equation (3.32).

In the case of $1 < l < p^m - 1$, α^l can always be represented in primal basis. Thus, $a\alpha^l$ has the same polynomial form as that in Equation (3.27). The following observation on the contents of the shift-register of $a\alpha^i$ will be helpful for the development of the dual basis multipliers. the contents of the registers are represented in vector form to have a better view of their relationships, where

$$\begin{array}{llllllll}
a & = [a_0' & a_1' & a_2' & \cdots & a_{m-2}' & a_{m-1}' \,] \\
a\alpha & = [a_1' & a_2' & a_3' & \cdots & a_{m-1}' & T_0' \,] \\
x\alpha^2 & = [a_2' & a_3' & a_4' & \cdots & T_0' & T_1' \,] \\
 & & & & \vdots & & \\
a\alpha^{m-1} & = [a_{m-1}' & T_0' & T_1' & \cdots & T_{m-1}' & T_{m-2}' \,],
\end{array}$$

and

$$T_i' = (a\alpha^{i+1})_{m-1}' .$$

By regarding the piled-up vectors as a matrix, it is shown to be symmetrical. This leads to a conclusion that the ith components $(a\alpha^l)_i'$ of the vector $a\alpha^l$ which are obtained by summing the ith component of the rows of the matrix can also be obtained by summing the columns of ith row of the matrix. Based upon the observation, the architecture of the dual basis multipliers are developed as follows.

The first design approach is based on that $a\alpha^l$ can be obtained by summing of the rows of the matrix and the algorithm which is developed in Algorithm 3.3. The input a is simultaneously loaded to the register at the initial clock. The register has feedback digit and is updated at clock i and the component of $a\alpha^l$ are accumulated at clock i+1. All the component of $a\alpha^l$ are simultaneously available at nth clock. A generic dual basis accumulated $a\alpha^l$ multiplier is shown in Figure 3.9. The architecture of this design is regular but need more hard-

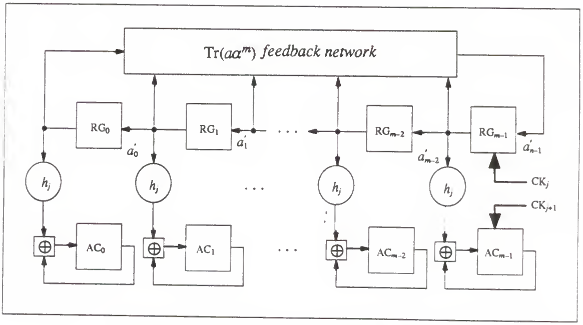


Figure 3.9 A Dual Basis Accumulated $a\alpha^l$ -Multiplier

ware. The advantage of this design is high speed processing capability in the output stage due to the single level delay which is independent to the number j. Thus, the system clock cycle is depend solely on the feedback network. The second design algorithm is based on the observation that $x\alpha^l$ can be obtained by summing the columns of the matrix.

Algorithm 3.5 Dual Basis Summed Multiplication.

INPUT: $[\ a_0,\ a_1,\ \ldots\ a_{m-2},\ a_{m-1}\ ]$ (Parallel–In)

$[\ h_0,\ h_1,\ \ldots\ h_{m-2},\ h_{m-1}\ ]$ (Parallel–In)

0. Initialization:

$$\mathrm{RG}_i \leftarrow a_i, \text{ for } i = 0, 1, \ldots, m-1;$$

$$(a\alpha^l)'_0 = h_0a'_0 + h_1a'_1 + \ldots + h_{m-2}a'_{m-2} + h_{m-1}a'_{m-1}\ ,$$

1. Clock 1: $\mathrm{RG}_i \leftarrow a\alpha$,

$$(a\alpha^l)'_1 = h_0a'_1 + h_1a'_2 + \ldots + h_{m-2}a'_{m-1} + h_{m-1}T'_0\ ,$$

2. Clock 2: $\mathrm{RG}_i \leftarrow a\alpha^2$,

$$(a\alpha^l)'_2 = h_0a'_2 + h_1a'_3 + \ldots + h_{m-2}T'_0 + h_{m-1}T'_1\ ,$$

$$\vdots$$

$m-1$. Clock $m-1$:

$$\mathrm{RG}_i \leftarrow a\alpha^{m-1}\ ,$$

$$(a\alpha^l)'_{m-1} = h_0a'_{m-1} + h_1T'_0 + \ldots + h_{m-2}T'_{m-1} + h_{m-1}T'_{m-2}\ ,$$

OUTPUT: $(a\alpha^l)'_i$ is sequentially available along the clock i (Serial–Out). ♦

In this design approach, a generic dual basis multiplier is depicted in Figure 3.10. The components a'_i of a is simultaneously loaded to the register at the first clock, then the components of $a\alpha^l$ are obtained sequentially. For instance, at clock i the ith component of $a\alpha^l$ is obtained by summing the current contents of the register by a d level adder tree adder (ADD). The level d in this case is equal to $\lceil \log j \rceil$, j is the number of nonzero h_i, and the number of adders required to implement ADD is therefore $j-1$. Combined the delay factor in the feed-

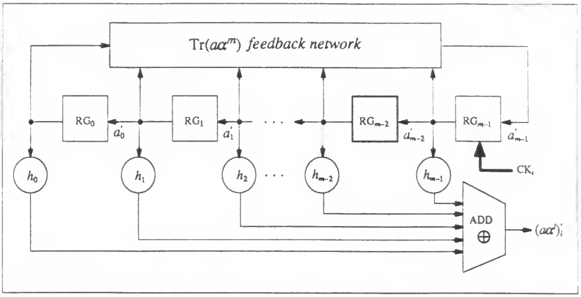


Figure 3.10 A Dual Basis Summed $a\alpha^l$ -Multiplier

back network, the system clock cycle is decided by the greater delay level of the feedback network and ADD. This architecture is simple and less hardware. It is suitable for sequential digit applications. However, the *d* level delay will significantly slow the system clock as *j* becomes large.

Both $a\alpha^l$ multipliers discussed above can be used to calculate the product of *a* and *b* where $a, b \in \mathrm{GF}(p^m)$. *a* is represented in dual basis as shown in Equation (3.28), while *b* is represented in primal basis as

$$b = h_0 + h_1\alpha + \ldots + h_{m-2}\alpha^{m-2} + h_{m-1}\alpha^{m-1},$$

then

$$ab = ah_0 + ah_1\alpha + \ldots + ah_{m-2}\alpha^{m-2} + ah_{m-1}\alpha^{m-1}.$$

This is exactly the form of Equation (3.27). Thus, a *ab*-multiplier can be realized by applying the same design in Figure 3.9 or 3.10 that *a* is represented in dual basis, *b* is in primal basis and the product is represented in dual basis form. Another implementation of *ab*-multiplier is similar to the primal basis nested multiplication algorithm in Algorithm 4.2. A generic dual basis *ab* multiplier is also developed and shown in Figure 3.11. The architecture of this de-

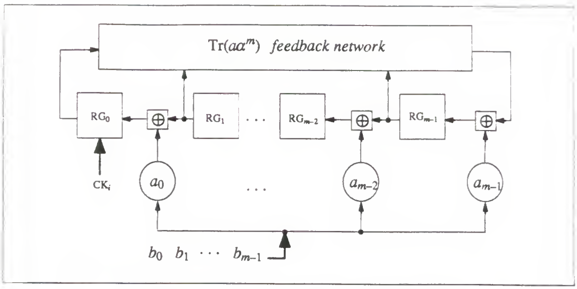


Figure 3.11 A Dual Basis Nested *ab* Multiplier

sign is simpler and less hardware than those shown in Figures 3.9 and 3.10. Besides the processing clock cycle totally depends on the feedback network.

It is unfortunate that two different bases are involved in the system since to actually use it as part of a larger device it would in general be necessary to have circuitry to change basis. However, for some bases, there exist some kind of self-dual properties, under such properties, the bases change is not thing but permutation of coordinates [McE87].

3.7 Normal Basis Multiplier

Massey and Omura [Wan85] invented a multiplier which performs the product of two elements in the finite field GF(2^m). In the normal basis representation the exponentiation by powers of p of an element in GF(p^m) is readily shown to be a simple cyclic shift of its digits. Multiplication in the normal basis representations requires for any one product digit the same logic circuitry as it does for any other product digit.

It is well know [Mac77] that a normal basis N_n exists in any finite field GF(p^m). Let $\beta \in$ GF(p^m), then one can find a normal basis N_n such that β is uniquely expressed as

$$\beta = b_0\alpha + b_1\alpha^p + b_2\alpha^{p^2} + \dots + b_{m-1}\alpha^{p^{m-1}},$$

where $b_i \in \mathrm{GF}(p^m)$, $i = 0, 1, \cdots, m-1$. According to the binomial theorem and the Fermat theorem that

$$\alpha^{p^m-1} \equiv 1 \bmod p^m$$

and

$$b_i^{p-1} \equiv 1 \bmod p$$

the pth power of β becomes

$$\beta^p = b_0^p\alpha^p + b_1^p\alpha^{p^2} + b_2^p\alpha^{p^3} + \dots + b_{m-1}^p\alpha^{p^m}$$

$$= b_{n-1}\alpha + b_0\alpha^p + b_1\alpha^{p^2} + \dots + b_{m-2}\alpha^{p^{m-1}} .$$

Hence, elements in $\mathrm{GF}(p^m)$ raised to powers of p can simply be realized by logic circuitry which accomplishes cyclic shift in a register. A cyclic shift register for power forming in finite fields is shown in Figure 3.12.

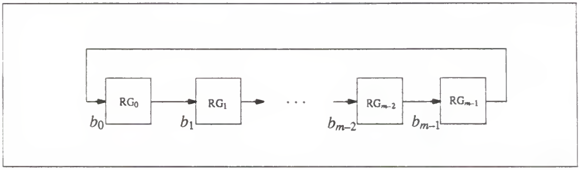


Figure 3.12 A Cyclic Shift Register for Power Forming in GF (p^m)

Let vectors $\beta = [\ b_0,\ b_1,\ \dots,\ b_{m-1}\ ]$ and $\gamma = [\ c_0,\ c_1,\ \dots,\ c_{m-1}\ ]$ be two elements of $\mathrm{GF}(p^m)$ in a normal basis representation. Hence, the last term d_{m-1} of the product

$$\delta = \beta\gamma = [\ d_0,\ d_1,\ \dots,\ d_{m-1}\ ]$$

is some function of the components of β and γ which can be defined as F_{NB} function. Therefore,

$$d_{m-1} = F_{NB}(\, b_0,\ b_1,\ \ldots,\ b_{m-1}\ ;\ c_0,\ c_1,\ \ldots,\ c_{m-1}\,). \qquad (3.33)$$

Since the exponential by powers of p means a cyclic shift of an element in a normal basis representation, one has

$$\begin{aligned} \delta^p &= \beta^p\ \gamma^p \\ &= [\ b_{m-1},\ b_0,\ \ldots,\ b_{m-2}\ ]\ [\ c_{n-1},\ c_0,\ \ldots,\ c_{n-2}\ ] \\ &= [\ d_{m-1},\ d_0,\ \ldots,\ d_{m-2}\ ]\,. \end{aligned}$$

Thus, the last component d_{m-2} of δ^p is obtained by the same F_{NB} function in Equation (3.33) operating on the components of β^p and γ^p, that is

$$d_{m-2} = F_{NB}(\, b_{m-1},\ b_0,\ \ldots,\ b_{m-2}\ ;\ c_{m-1},\ c_0,\ \ldots,\ c_{m-2}\,).$$

By repeating above procedure, the product operation simply requires one logic function F_{NB} of $2m$ component of β and γ to sequentially compute the m components of the product. Figure 3.13 illustrates the general logic diagram of the normal basis multiplier where the gate

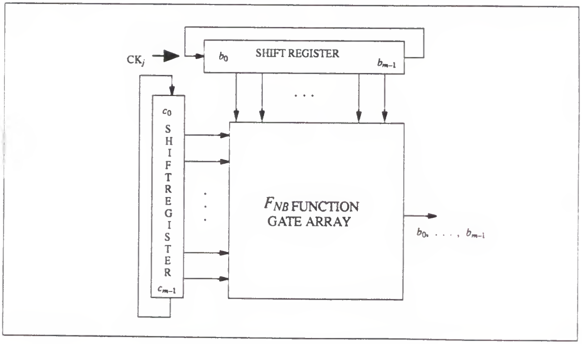


Figure 3.13 A Normal Basis Multiplier

array is a VLSI implementation of F_{NB} function.

CHAPTER 4
FINITE STRUCTURE TRANSFORMS

4.1 Introduction

Finite digital convolution is a numerical procedure which has many very powerful applications. It is used to implement finite impulse response (FIR) and infinite impulse response (IIR) digital filters; to carry out auto and cross correlation; as well as to perform the computations such as polynomial multiplication and multiplication of very large integers [Aga75]. There are several methods to implement finite convolution that differ in the amount of computation required, the effects of arithmetic round-off, and the amount of storage required. It is somewhat difficult to compare various algorithms because of the trade-offs in hardware and software implementations. However, because of the complexity of performing multiplication, the number of multiplications necessary to implement convolution is often an important factor to be minimized.

The use of the cyclic convolution property (CCP) of discrete Fourier transforms (DFT's) can reduce the computational complexity of finite convolution, only when some fast Fourier transform (FFT) algorithms are available and applied to the DFT's. However, the major disadvantage of this approach is in the form of significant amounts of round-off error due to the accumulated operation in the transform when consider the transform performing in complex number field, **C**.

Recently, various researchers have proposed the use of transforms over finite structures using number theoretic concepts for error-free, fast, efficient computations of finite digital convolutions of real integer sequences or complex integer sequences. The implementation of these transforms involve the use of modular arithmetic (as discussed in Chapter 2) and often depend on the choice of the modulus, M, a large class of transforms exist that have the CCP. By special choices of the length of sequence, N, the modulus M, and the value of the transform factor, r, it is possible to have transforms that need only word shifts and additions but no multiplications, that have an FFT-type fast algorithm, that do not require storage of complex value r, and that have no round-off errors. These transforms are called number theorem transforms (NTT's). In particular, the transform with a modulus of the form of tth Fermat number, F_t, referred to as the Fermat number transform (FNT), has been described in detail by Agarwal and Burrus [Aga74a], [Aga75]. A Mersennne number transform (MNT) with modulus $M = 2^p - 1$, p prime, has been defined by Rader [Rad72]. The main disadvantage of both the FNT and MNT is the requirement of a rigid relationship between the dynamic range and obtainable transform length. There is also a limited choice of possible wordlengths.

Transforms in finite structures also are applicable over extension fields. During the past few years there has been strong interest among researchers [Ree75] in transforms over the second order extension field, GF(p^2), which is analogous to those used for the complex field C. The use of these transforms offer a number of advantages. In many applications, such as in radar or communication, the sequences to be convolved consist of complex quantities; the transform also allows greatly increased sample lengths over those defined in the ground field, GF(p), like those of FNT or MNT; for a sufficiently large p, one can use this transform to convert a sequence of complex integers a_n into the sequence A_k in GF(p^2) for which the inverse transform of A_k is precisely the original sequence of complex numbers a_n. Consequently, filtering operations or convolutions without round-off error are obtained using this transform on a sequence of complex integers.

For the higher order extension field, GF(p^m), transforms exist as long as the sufficient and necessary conditions exist. By applying finite field properties, such as the conjugacy property, and the basis representation of the field elements to the existing fast transform algorithms. A significant reduction in computational complexity is achieved. There also has been considerable work on high-speed residue number arithmetic for use in high data rate digital signal processing. The structure of a complex integer ring is generalized to form a complex residue number system (CRNS), which is a formidable computational number system, independent of any relation it may have to the NTT's. In this CRNS, complex multiplication is accomplished by means of a real index calculus (discrete logarithm) because the selection of the system parameters is not overly constrained by the algebraic structure required by a NTT. The CRNS is only useful when the complex variables are feasibly represented by the power form over a second order extension field and computed using the discrete logarithm methods [Jen80a]. When complex variables are represented by a isomorphic mapping, traditionally referred to as the quadratic residue number system (QRNS), the usual operations are reduced to a point-wise operation.

4.2 Properties of Transforms and Some Prime Field Transforms

Define

$$O(M) = \gcd(p_1 - 1,\ p_2 - 1, \cdots,\ p_l - 1). \tag{4.1}$$

Note that $N \mid \gcd(p_1 - 1,\ p_2 - 1, \cdots,\ p_l - 1)$, so that a necessary and sufficient condition for the existence of an N-point NTT is that

$$N \mid O(M). \tag{4.2}$$

In practice it is often easier to verify the following three necessary and sufficient conditions for the existence of N-point NTT defined modulo a composite number M:

1) $\alpha^N \equiv 1 \bmod M$,

2) $NN^{-1} \equiv 1 \bmod M$,

3) $\gcd(\alpha^l - 1, M) = 1$ for all l such that N/l is a prime number.

As an example, let $M = p^2 = 3^2$ and $\alpha = 8$. Then 8 is a root of order 2 modulo 9, $N = 2$, $\gcd(M, N) = 1$, $N^{-1} \equiv 5 \bmod 9$, and $N \mid (p - 1)$.

Cyclic Convolution Property (CCP). If $g(n)$ and $h(n)$, $n = 0, 1, 2, \ldots, N-1$, are two periodic sequences with period N, their cyclic convolution is a periodic sequence $a(i)$, $i = 0, 1, 2, \ldots, N-1$, with period N described by

$$a(i) = \sum_{n=0}^{N-1} h(i-n)g(n) . \tag{4.3}$$

If the discrete transforms T of the sequences $g(n)$, $h(n)$, and $a(n)$ can be related as

$$T[\, a(n)\,] = T[\, h(n)\,]\, T[\, g(n)\,], \tag{4.4}$$

then the transform has the cyclic convolution property (CCP). Hence the CCP states that the transform of the cyclic convolution of two sequences is the product of the transforms of the two sequences. Certainly, the DFT has this property and that

$$a(n) = \mathrm{IDFT}\{\, \mathrm{DFT}[\, h(n)\,]\, \mathrm{DFT}[\, g(n)\,]\, \}. \tag{4.5}$$

DFT Structure. If an N-point sequence $x(n)$ and its transform $X(k)$ can be related by a transform pair

$$X(k) = \sum_{n=0}^{N-1} x(n)\alpha^{nk} , \quad k = 0, 1, \ldots, N-1, \tag{4.6}$$

and

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\alpha^{-nk} , \quad n = 0, 1, \ldots, N-1, \tag{4.7}$$

then the transform whose basis functions are α^{nk} is said to have a DFT structure. In this case both the forward and inverse transforms have similar operations. In Equation (4.7), N^{-1} represents the multiplicative inverse of N in the field in which the arithmetic is carried out. An N-point transform having the DFT structures has the CCP — provided N^{-1} exists, and α is a primitive root of order N. When all the transform operations are carried out in a field of integers modulo M, the transform belongs to the NTT. In a ring of integers $\mathbf{Z}_M$, as $-k \equiv M-k \bmod M$, conventional integers can be uniquely represented only if their absolute value is less than

$M/2$. Since the convolution is implemented in modular arithmetic, so long as the magnitude of the convolution of two sequences does not exceed $M/2$, the NTT can yield the same result as that obtained using conventional arithmetic.

Although there are a large class of NTT that can implement cyclic convolution, only a few of them are computationally efficient when compared to the DFT and other techniques. Three constraints dictate the selection of NTT for discrete convolution:

1) N should be highly composite so that the NTT may have a fast algorithm, and it should be large enough for application to long sequence lengths;

2) Multiplication by powers of α should be a simple operation. If α and its powers have a simple binary representation, then this multiplication reduces to bit shifting;

3) To simplify modular arithmetic, M should have property 2) and should be large enough to prevent overflow;

4) Another constraint on the NTT is that the word length of the arithmetic be related to the maximum length of the sequence. For example, for the FNT, when $\alpha = \sqrt{2}$, $N = 2^{t+2} = 4b = 4$ times the word length. When $\alpha = 2$, $N = 2^{t+1} = 2b = 2$ times the wordlength. This constraint can, however, be minimized by adopting multidimensional techniques for implementing one-dimensional convolution [Aga74a].

Selection of the modulus M, sequence length N, and the order of α modulo M is based on meeting the above constraints so that the efficient NTT can be developed. For example, if M is even, then by [Aga74b] the maximum possible sequence length is 1, a case of no interest. when M is a prime number, $N_{\max} = M - 1$. Finally, when $M = 2^k - 1$ and k is a composite number $k = PQ$, where P is a prime number and Q is not necessarily a prime number, then

$$(2^P - 1) \mid (2^{PQ} - 1), \tag{4.8}$$

and $N_{\max} = 2^P - 1$.

Fermat Number Transform (FNT). If $M = 2^k + 1$ and k is odd, then $3 \mid (2^k + 1)$. Hence $N_{\max} = 2$. When k is even and $k = s2^t$, where s is an odd integer and t is an integer,

$$(\ 2^{2^t} + 1) \mid (\ 2^{s2^t} + 1\ ), \tag{4.9}$$

and the sequence length is governed by $2^{2^t} + 1$. For integers of the form $M = 2^{2^t} + 1$, called Fermat numbers, the NTT reduces to the Fermat number transform (FNT). F_t is the tth Fermat number, defined as

$$F_t = M = 2^{2^t} + 1. \tag{4.10}$$

Of all the Fermat numbers only F_0 to F_4 are prime. The FNT and its inverse can be defined as

$$X_f(k) = [\ \sum_{n=0}^{N-1} x(n)\alpha^{nk}\ ] \bmod F_t\,, \quad k = 0, 1, \ldots, N-1, \tag{4.11}$$

$$x(n) = [\ N^{-1} \sum_{k=0}^{N-1} X_f(k)\alpha^{-nk}\ ] \bmod F_t\,, \quad n = 0, 1, \ldots, N-1, \tag{4.12}$$

where N is the order of α modulo F_t such that $\alpha^N \equiv 1 \bmod F_t$.

Mersenne Number Transform (MNT). Mersenne number are the integers given by $2^P - 1$ where P is prime. When M is a Mersenne number the NTT is called the Mersenne number transform (MNT). When $\alpha = -2, N = N_{\max} = 2P$. When $\alpha = 2, N = P$, since $2^P = M + 1 \equiv 1 \bmod M$. Mersenne numbers, denoted here M_p, are 1, 3, 7, 31, 127, 2047, 8191, ... For $\alpha = 2$ the MNT and its inverse can be defined respectively as

$$X_m(k) = [\ \sum_{n=0}^{P-1} x(n)2^{nk}\ ] \bmod M_P\,, \quad k = 0, 1, \ldots, P-1, \tag{4.13}$$

$$x(n) = [\ P^{-1} \sum_{k=0}^{P-1} X_m(k)2^{-nk}\ ] \bmod M_P\,, \quad n = 0, 1, \ldots, P-1, \tag{4.14}$$

where $P^{-1} = M_p - (M_p - 1) / P$.

Rader [Rad72] has shown that the MNT satisfies the CCP and has discussed hardware implementation for the MNT.

Rader Transform (RT). The Rader transform is a special case of the NTT. For any Fermat number, 2 is of order $N = 2b = 2^{t+1}$; that is, $2^{2b} \equiv 1 \bmod F_t$. When α is any power of 2 all the multiplications by α^{nk} become bit shifts and the FNT can be computed very efficiently; for the case $\alpha = 2$, both the FNT and MNT are called the Rader transforms. When N is an integer power of 2, the Rader transform can be implemented by a radix-2 FFT-type algorithm. Substituting 2 for the multiplier $w = \exp(-j2\pi/N)$ in the FFT flowgraph yields the fast algorithm for the Rader transform.

4.3 Complex Fields Transforms and Complex Arithmetic

A multitude of important signal processing and communication applications such as the computation of FFT and auto-correlation or cross-correlation of multiphase communication signals require complex transforms and a large number of complex multiplications. Thus, the NTT in complex field and a low complexity complex number multiplication scheme are introduced in the following sections.

Digital filtering of complex signals or cyclic convolution of complex sequences can be accomplished in a complex integer field. In a ring of complex integers, $\mathbf{Z}_M^c$, all the arithmetic operations are performed as in normal complex arithmetic except that both the real and imaginary parts are evaluated separately mod M. The set $c_j = a_j + ib_j,\ a_j, b_j = 0, 1, \ldots, M-1$, where $a_j = \mathrm{Re}[c_j\ ]$ and $b_j = \mathrm{Im}[c_j\ ]$, represents $\mathbf{Z}_M^c$. All complex integers are congruent modulo M to some complex integer in this set. Complex convolutions arise in many fields, such as radar, sonar, and modem equalizers.

4.3.1 Complex Number Theoretic Transform (CNNT).

There are two distinct situations, which are handled quite differently, depending on whether or not the square root of –1, denoted by $\sqrt{-1}$, exists in GF(p). If p is a Mersenne prime, then $\sqrt{-1}$ does not exist in GF(p); if p is a Fermat prime, then $\sqrt{-1}$ does exist in GF(p).

Complex MNT. For $p = 2^m - 1$, a Mersenne prime M_p, where m is an odd prime, $\sqrt{-1}$ does not exist in this field GF(M_p). Thus, we extend the field to GF(M_p^2) in the same way that the real field **R** is extended to the complex number field **C**. The Galois field Fourier transform over GF(M_p^2) can used to compute convolutions in the complex field.

In the Galois field GF(M_p), the polynomial $x^2 + 1$ has no zeros. Hence, the field is extended by adjoining an element called j and forming the set GF(M_p^2) $=$ $F_{2^m-1}(j) = \{ a + jb \}$, where a and b are elements of GF(M_p).

As shown in Equations (4.13) and (4.14), the MNT transform pair can be utilized in the complex number case except that the input sequence length d is $M_p^2 - 1$ or a divisor thereof; the transform factor α is an element of GF(M_p^2) of order d. By looking more detail into the length d that we have the factorization

$$M_p^2 - 1 = (2^m - 1)^2 - 1 = 2^{m+1}(2^{m-1} - 1)$$

Hence, in GF(M_p^2) we can choose 2^{m+1} or some factors of $2^{m-1} - 1$ as the data length of the transform. In the case of d is a factor of two, the transform can be computed by a radix-two Cooley-Tukey FFT. This transform have more choices for the data length than in MNT within the finite field GF(M_p).

Complex FNT. In the case of p a Fermat number, F_t , it is not possible to form an extension field GF(p^2) with a multiplication rule that behaves like complex multiplication since $\sqrt{-1}$ is an element of GF(p). Specifically, $\sqrt{-1} = 2^{m/4}(2^{m/2} - 1)$.

4.3.2 Complex Residue Number System (CRNS)

The execution of very high speed complex arithmetic is important in spectral analysis and in the processing of complex baseband waveforms that result from quadratic demodulation in radar and communication systems. For example, modern synthetic aperture radars operating in the spotlight mode require polar-to-rectangular format interpolation filters that can operate in real time as an aircraft flies by the scene. These interpolation filters are ideally spatially varying complex FIR filters of relatively low order. It appears that this type of requirement is well suited for a complex RNS realization.

If the field GF(p) does not contain a square root of –1, then GF(p) can be extended to GF(p^2) in the same way that the real field is extended to the complex number field. For example, $6 \equiv -1 \bmod 7$ in GF(7), and it is easy to check that there is no element in GF(7) whose square is 6. Hence, we define

$$\mathrm{GF}(7^2) = \mathrm{GF}(49) = \{ a + jb : a, b \in \mathrm{GF}(7) \}$$

with addition and multiplication defined in the same way as for the complex number field.

$$[(a + jb) + (c + jd)] \bmod p = [(a + c) + j(b + d)] \bmod p \qquad (4.15)$$

$$[(a + jb) \cdot (c + jd)] \bmod p = [(ac - bd) + j(ab + cd)] \bmod p. \qquad (4.16)$$

With these definitions, GF(p^2) is a field. The integer of GF(p^2) are the elements of GF(p), so GF(p^2) has characteristic p.

4.3.3 Quadratic Residue Number System (QRNS)

In a conventional residue number system, such as CRNS, a complex multiplication requires four real multiplications and two real additions per moduli. The quadratic residue number system (QRNS) changes this requirement significantly. This system is defined with an isomorphic mapping originally suggested in the literature by Vanwormhoudt [Van78] and later on by Leung [Leu81] and Krogmeier and Jenkins [Kro83].

In a prime field GF(p), those elements that have a square root are called quadratic residues (because they are the squares of their square roots modulo p). Exactly half of the nonzero elements in GF(p), p an odd prime, have square roots. To see this (refer to Table 2.3), first note that every even power of a primitive element α has a square root. On the other hand, every element that is a square root can be written as α^n for some n, and so its square is $\alpha^{2n \bmod p-1}$, since the multiplicative group of the field is cyclic with $p-1$ elements. But $p-1$ is even, so $(2n) \bmod (p-1)$ is even as well. Hence only even powers of α can have square roots. For the finite field GF(17) in Table 2.3, there are 16 nonzero elements. Eight of these elements are quadratic residue, which have square roots. They are the even power of the primitive element 3, which are 9, 13, 15, 16, 8, 4, 2, 1. For the application in complex fields, we concentrate on the case of the quadratic residue -1 or $p-1$.

Consider a modulus channel with modulus M and a complex integer $z = x + iy$ of $\mathbf{Z}_M^c$, where $x, y \in \mathbf{Z}_M$, $i = \sqrt{-1}$. Then the quadratic RNS mapping is an isomorphic mapping F_2 of $\mathbf{Z}_M^c$ onto the external direct product $\mathbf{Z}_M \times \mathbf{Z}_M = \mathbf{Z}_M^2$. This isomorphic mapping satisfies the following equation,

$$z = x + iy \underset{F_2}{\overset{F_2^{-1}}{\leftrightarrows}} (Z_0\ ,\ Z_1\ ). \tag{4.17}$$

The input and output parameters x, y and Z_0, Z_1 satisfy

$$Z_0 = (x + jy) \bmod M,$$

$$Z_1 = (x - jy) \bmod M; \tag{4.18}$$

$$x = (2^{-1}(Z_0 + Z_1)) \bmod M,$$

$$y = (2^{-1} j^{-1}(Z_0 - Z_1)) \bmod M. \tag{4.19}$$

where x, y and Z_0, $Z_1 \in \mathbf{Z}_M$, j is a quadratic root of -1 in $\mathbf{Z}_M$, 2^{-1} and j^{-1} are the multiplicative inverses of 2 and j mod M, respectively. The structure $\mathbf{Z}_M^2$ is called the QRNS and is a finite ring consisting of M^2 elements.

The rules of composition in $\mathbf{Z}_M^2$ are as follows.

1) Addition:

$$(Z_{00}, Z_{01}) + (Z_{10}, Z_{11}) = [(Z_{00} + Z_{10}) \bmod M, (Z_{01} + Z_{11}) \bmod M]. \tag{4.20}$$

2) Multiplication:

$$(Z_{00}, Z_{01}) (Z_{10}, Z_{11}) = [(Z_{00} \cdot Z_{10}) \bmod M, (Z_{01} \cdot Z_{11}) \bmod M]. \tag{4.21}$$

From these rules, we see that the multiplication in the new domain is simply a componentwise operation.

Again, for the QRNS mapping F_2 to exist, the quadratic congruence, $x^2 \equiv -1 \bmod M$, must be solvable. The necessary and sufficient condition for the quadratic roots of -1 in $\mathbf{Z}_M$ is that $M = 4k + 1$, where k is a positive integer. Furthermore, two roots, j and j^*, are mutually additive and multiplicative inverses mod M, that is $j^* \equiv -j \equiv j^{-1} \bmod M$ [Har65, Lip81].

4.3.4 Extended Algebraic Integer and Polynomial Residue Number System (PRNS)

The principle advantage of RNS processing is its ability to reduce a complex multiplication or addition to the calculation of a single integer multiplication or integer addition modulo a prime in parallel sets of residue channels. When the primes are small the implementation complexity of the computation in each of the residue channels is correspondingly small — implying substantially high throughput.

To work in an RNS the complex roots of unity needed in the computation, and also the input data, must first be quantized. This quantization is conventionally performed by scaling and then rounding to the nearest Gaussian integer. This method introduces errors that are dependent on the scale factor. The large scale factors that constrain the quantization error also impose a large dynamic range requirement on the RNS, thus offsetting its advantage.

Instead, Cozzens and Finkelstein [Coz85] introduced an idea of performing the computation in a certain ring of algebraic integers that contain the Gaussian integers. This is accomplished by approximating the appropriate complex roots of unity by elements of the ring. Games [Gam85] illustrated a method to perform this approximation by elements of the algebraic integers of $\mathbf{Q}(\omega)$. Due to the nature of these approximations, the dynamic range of the computation is dramatically reduced.

Algebraic Integer Concepts. To reduce the range requirements, Gaussian integers $u + jv$, with u and v small, can be used to approximate only the arguments of the nth roots of unity. The approach of Gaussian integers, which are the algebraic integers of the field $\mathbf{Q}(i)$, was replaced by approximations using algebraic integers in higher degree extensions of $\mathbf{Q}$. Typically, the role of $i = \exp(j2\pi/4)$, a primitive fourth root of unity, is replaced by $\omega = \exp(j2\pi/8)$ or $\omega = \exp(j2\pi/16)$. while algebraically similar, the algebraic integer $\mathbf{Z}[\omega] \subseteq \mathbf{Q}(\omega)$ can differ from $\mathbf{Z}[i]$ dramatically in one sense that $\mathbf{Z}[\omega]$ can be dense in $\mathbf{C}$ so that arbitrarily good approximations can be obtained.

Let ω be a primitive nth root of unity in $\mathbf{C}$ with $n > 2$, for instance $\omega = \exp(j2\pi/n)$. It is known [Niv80, Chapter 2] that ω satisfies an irreducible monic polynomial $C_n(x)$ of degree $\phi(n)$, and this cyclotomic polynomial is

$$C_n(x) = \prod_i (x - \omega^i), \tag{4.22}$$

where the product is taken over positive integers $i \le n$ that are relative prime to n. Similar to the concept of constructing finite extension fields that discussed in Chapter 3, the subfield of $\mathbf{C}$ obtained by adjoining ω to the rational numbers $\mathbf{Q}$, denoted by $\mathbf{Q}(\omega)$, has , as a vector space over $\mathbf{Q}$, dimension $\phi(n)$. In fact,

$$\mathbf{Q}(\omega) = \{ a_0 + a_1\omega + \ldots + a_{\phi(n)-1}\omega^{\phi(n)-1} : a_i \in \mathbf{Q} \}. \tag{4.23}$$

Since we deal with finite structures in this dissertation, we concern with the subset $\mathbf{Z}[\omega] \subseteq \mathbf{Q}(\omega)$ defined by

$$\mathbf{Z}[\omega] = \{ z_0 + z_1\omega + \ldots + z_{\phi(n)-1}\omega^{\phi(n)-1} \ : z_i \in \mathbf{Z} \}. \tag{4.24}$$

$\mathbf{Z}[\omega]$ forms a ring under complex addition and multiplication, where the rule implied by $C_n(\omega) = 0$ is used to reduce powers of ω larger than $\phi(n) - 1$. Also any element $z \in \mathbf{Z}[\omega]$ is a root of a monic polynomial (minimal polynomial) $x^m + c_{m-1}x^{m-1} + \ldots + c_0$ with coefficients $c_i \in \mathbf{Z}$, and degree $m \geq 1$. Furthermore, $\mathbf{Z}[\omega]$ is exactly the subset of $\mathbf{Q}(\omega)$ that has elements with this property. As such, $\mathbf{Z}[\omega]$ is called the ring of algebraic integers of $\mathbf{Q}(\omega)$. It is well known that when $n = 4$, $\mathbf{Z}[\omega]$ is just the Gaussian integers.

If $z = x + jy = z_0 + z_1\omega + \ldots + z_{\phi(n)-1}\omega^{\phi(n)-1} \in \mathbf{Z}[\omega]$, then z is represented either by its complex coordinates (x, y), thought of as a point in $\mathbf{R}^2$, or by $\mathbf{Z}[\omega]$ coordinates $[z_0, z_1, \ldots, z_{\phi(n)-1}]$. The rules of composition of two elements u and v of $\mathbf{Z}[\omega]$ when added yield $u + v = [u_0 + v_0, u_1 + v_1, \ldots, u_{\phi(n)-1} + v_{\phi(n)-1}]$. If $m \in \mathbf{Z}$, then $mu = [mu_0, mu_1, \ldots, mu_{\phi(n)-1}]$. With multiplication defined by

$$uv = c = \sum_{j=0}^{\phi(n)-1} c_j\omega^j \tag{4.25}$$

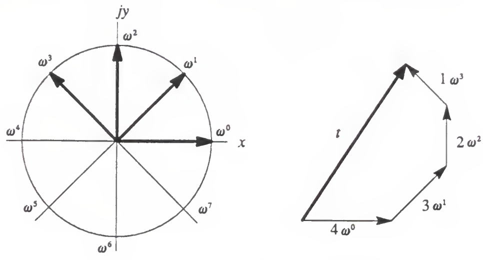
where

$$c_k = \sum_{j=0}^{k} a_{k-j}b_j - \sum_{j=k+1}^{\phi(n)-1} a_j b_{\phi(n)-j+k} .$$

Observe that multiplication in $\mathbf{Z}[\omega]$ is similar to the circular convolution of the two sequences u and v.

<u>Algebraic Integer Extension Approximations.</u> In this presentation we demonstrate a system based on a complex extension of degree 4. Let $\omega = \exp(j2\pi/8)$ be the primitive eighth root of unity. Then ω satisfies the equation, generated by the cyclotomic polynomial

$C_8(x)$, $\omega^4 + 1 = 0$. The cyclotomic fields $\mathbf{Q}(\omega)$ has the form of $\mathbf{Q}(\omega) = \{ a_0 + a_1\omega^1 + a_2\omega^2 + a_3\omega^3 : a_i \in \mathbf{Q} \}$. Its respective ring of algebraic integers, is defined by $\mathbf{Z}[\omega] = \{ z_0 + z_1\omega^1 + z_2\omega^2 + z_3\omega^3 : z_i \in \mathbf{Z} \}$. An element z of $\mathbf{Z}[\omega]$ is represented by $[z_0, z_1, z_2, z_3]$. The representation of a complex number t by $\mathbf{Z}[\omega]$ coordinate is demonstrated in Figure 4.1.



$$\begin{aligned} t &= x + iy = 4\omega^0 + 3\omega^1 + 2\omega^2 + 1\omega^3 \\ &= 5.4142 + i4.82843 \\ &= [\ 4,\ 3,\ 2,\ 1\ ] \end{aligned}$$

Figure 4.1 The Representation of Complex Number t in $\mathbf{Z}[\omega]$

For an arbitrary complex number $t = x + jy$, where $x, y \in \mathbf{R}$, an algebraic integer s can be chosen to approximate t. Since $\omega = (\sqrt{2}/2) + i(\sqrt{2}/2)$ such that

$$x = a_0 + \frac{\sqrt{2}}{2}(a_1 - a_3) = x_0 + x_1\sqrt{2},$$

$$y = a_2 + \frac{\sqrt{2}}{2}(a_1 + a_3) = y_0 + y_1\sqrt{2}$$

where $x_0 = a_0$, $y_0 = a_2$, $x_1 = (a_1 - a_3)/2$, and $y_1 = (a_1 + a_3)/2$.

It is obvious that the problem of approximation of complex numbers by an algebraic integer ω is equivalent to the approximation of real numbers **R** by $\sqrt{2}$. In fact, we propose the one-dimensional approximation to reduce the complexity of two-dimensional approximations, which suggested by Games [Gam85]. In the case of finding integers x_0 and x_1 to approximate $\sqrt{2}$, we define the approximation error $\epsilon = | \sqrt{2} - (x_0/x_1) |$, where x_0 and x_1 are integers. According to the mean value theorem, it follows that

$$\frac{f(x) - f(x_0/x_1)}{x - (x_0/x_1)} = f'(\eta) \tag{4.26}$$

for some $\eta \in [\sqrt{2}, (x_0/x_1)]$ and $f(x) = x^2 - 2$. Letting $x = \sqrt{2}$, the Equation (4.26) becomes

$$\sqrt{2} - (x_0/x_1) = -f(x_0/x_1)(f(\eta)^{-1}).$$

Therefore,

$$\epsilon = | \sqrt{2} - (x_0/x_1) | = | ((x_0^2/x_1^2) - 2)/(2\eta) |.$$

Thus, the approximation error ϵ becomes

$$\epsilon = (| (x_0^2 - 2x_1^2)/x_1^2 |) (| 1/(2\eta) |) \geq 1/(3x_1^2)$$

and also

$$\epsilon \leq (| (x_0^2 - 2x_1^2)/x_1^2 |) (1/2) \leq (2/x_1^2) (1/2) = (1/x_1^2).$$

Thus, there exists a range for the approximation error ϵ

$$1/(3x_1^2) \leq \epsilon \leq 1/x_1^2 . \tag{4.27}$$

Polynomial Residue Number Systems (PRNS). The ring of Gaussian integers modulo M, denoted as $\mathbf{Z}_M[i]$, forms the basis of QRNS with M of the form $4k+1$. It is easy to see that the product of two complex numbers is equivalent to the product of two first order polynomials taken module $(x^2 + 1)$. Extending the QRNS idea, we introduce the algebraic-integers of higher degree to achieve a better approximation to the complex numbers.

The product of two $(N-1)$-order polynomials modulo $x^N + 1$ over some modular ring $\mathbf{Z}_M$ define a residue number system of order N. To obtain the lowest possible multiplication counts within the prime field, the Nth order congruence, $x^N \equiv -1 \bmod M$, must be solvable. That is, the polynomial $x^N + 1$ must be factored in N distinct factors in $\mathbf{Z}_M$ as

$$x^N + 1 \equiv (z - r_0)\ (z - r_1)\ \ldots\ (z - r_{N-1}) \bmod M, \tag{4.28}$$

with $r_0, r_1, \ldots, r_{N-1} \in \mathbf{Z}_M$. The necessary and sufficient condition for the Nth order roots of -1 to exist in the integer ring $\mathbf{Z}_M$ is $N \mid (M-1)/2$ [Ska87, Lip81]. As a result, there exists an isomorphic mapping F_N of $z(x)$ of $\mathbf{Z}_M[\,i\,]$ onto the external direct product $\mathbf{Z}_M \times \mathbf{Z}_M \times \ldots \times \mathbf{Z}_M \equiv \mathbf{Z}_M^N$, where

$$z(x) = z_0 + z_1 x^1 + \ldots + z_{N-1} x^{N-1}, \tag{4.29}$$

with $x = \exp(j2\pi/N)$ and is a primitive Nth root of unity in $\mathbf{C}$, and $z_i \in \mathbf{Z}_M$, $i = 0, 1, \ldots, N-1$. This is called the Polynomial Residue Number System (PRNS). The PRNS isomorphic mapping satisfies the following relationship:

$$z(x) = z_0 + z_1 x^1 + \ldots + z_{N-1} x^{N-1} \ \underset{F_N}{\overset{F_N^{-1}}{\leftrightarrows}}\ (\,Z_0,\ Z_1,\ \ldots,\ Z_{N-1}\,)\,. \tag{4.30}$$

The input and output parameters z_i, Z_i satisfy the equations

$$Z_i = z(x) \bmod (x - r_i) = z(r_i) \bmod M \tag{4.31}$$

and

$$z(x) = \Big(\sum_{i=0}^{N-1} Z_i Q_i(x)\Big) \bmod (x^N + 1), \tag{4.32}$$

where z_i, Z_i, $r_i \in \mathbf{Z}_M$, $i = 0, 1, \ldots, N-1$, and

$$Q_i(x) = N^{-1}(\ 1 + r_i^{-1} x + r_i^{-2} x^2 + \ldots + r_i^{-(N-1)} x^{N-1}\ ).$$

The mapping also can be represented in vector-matrix forms as

$$Z^T = F_N \, z^T \tag{4.33}$$

and

$$z^T = F_N^{-1} Z^T , \tag{4.34}$$

where $z = [\, z_0 , z_1 , \ldots , z_{N-1} \,]$ and $Z = [\, Z_0, Z_1, \ldots , Z_{N-1} \,]$ are vectors. The matrix forms F_N and F_N^{-1} of the isomorphic mapping are

$$F_N = \begin{bmatrix} 1 & r_0 & r_0^2 & & r_0^{N-1} \\ 1 & r_1 & r_1^2 & & r_1^{N-1} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ 1 & r_{N-1} & r_{N-1}^2 & & r_{N-1}^{N-1} \end{bmatrix} \tag{4.35}$$

and

$$F_N^{-1} = N^{-1} \begin{bmatrix} 1 & 1 & & 1 \\ r_0^{-1} & r_1^{-1} & & r_{N-1}^{-1} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & & \cdot \\ r_0^{-(N-1)} & r_1^{-(N-1)} & & r_{N-1}^{-(N-1)} \end{bmatrix} . \tag{4.36}$$

The rules of composition in $\mathbf{Z}_M \times \; \mathbf{Z}_M \times \ldots \times \mathbf{Z}_M \equiv \mathbf{Z}_M^N$ are

1) addition:

$(Z_{00} , Z_{01} , \ldots , Z_{0N-1}) + (Z_{10} , Z_{11} , \ldots , Z_{1N-1})$

$$= [\, (\, Z_{00} \; + Z_{10}\,) \bmod M, (Z_{01} \; + Z_{11}\,) \bmod M, \ldots , (Z_{0N-1} \; + Z_{1N-1}\,) \bmod M \,]; \tag{4.37}$$

2) multiplication:

$(Z_{00} , Z_{01} , \ldots , Z_{0N-1}) \, (Z_{10} , Z_{11} , \ldots , Z_{1N-1})$

$$= [\, (\, Z_{00} \, Z_{10}\,) \bmod M, (Z_{01} \, Z_{11}\,) \bmod M, \ldots , (Z_{0N-1} \, Z_{1N-1}\,) \bmod M \,], \tag{4.38}$$

where $Z_{ij} \in \mathbf{Z}_M$. From these rules, multiplication in the new domain is simply a component-wise operation; and as such, it requires only N operations for an N-tuple data stream as opposed to N^2 operations necessitated by the conventional approach. This computational complexity reduction is encouraging. However, the main concern involves isomorphic mappings which are discussed in later chapter.

Since the complex arithmetic of residue number systems in DSP applications is a recent subject of intense study [Ree75b, Jen80, Kro83], the application PRNS in complex multiplication seems to be viable solution which offers a tremendously low complexity in complex operations. A possible complex number arithmetic model is shown in Figure 4.2 . In this model, analog signals are first digitized by analog-to-digital (A/D) converter and approximated by algebraic integers. PRNS mappings are then applied to the polynomial form of input signals. A low complexity component-wise operation is executed in the PRNS domain. The result is mapped back to the algebraic integer polynomial format followed by a digital-to-analog (D/A) converter.

4.4 Transforms and Computations in Extension Fields

All the finite field transforms and the RNS systems discussed performs over a ground field GF(p) where p may be a Fermat or Mersenne number, or p satisfies some specified condition such as having the form of $4k + 1$ or $2Nk + 1$. Without meeting these requirements, transforms may exist in the higher order extension field GF(p^m). A number of previous discussed finite field properties such as index calculus, conjugacy of field elements, and basis representation to perform and expedite these transforms in extension fields.

4.4.1 Index Calculus Complex Residue Number System (ICCRNS)

ICQRNS. For certain primes, the quadratic equation $x^2 \equiv -1 \bmod M$ does not have a solution in $\mathbf{Z}_M$. In this case -1 is called a quadratic nonresidue mod M. It is follows that a solution to the equation can be found in the second order extension field GF(M^2) (or $\mathbf{Z}_{M^2}$) and is denoted $\hat{j} = \sqrt{-1}$. Under these circumstances the set of elements in GF(M^2) is

$$\mathbf{Z}_{M^2} = \mathbf{Z}_M(\hat{j}) = \{ a + \hat{j}\, b : a, b \in \mathbf{Z}_M \}, \tag{4.39}$$

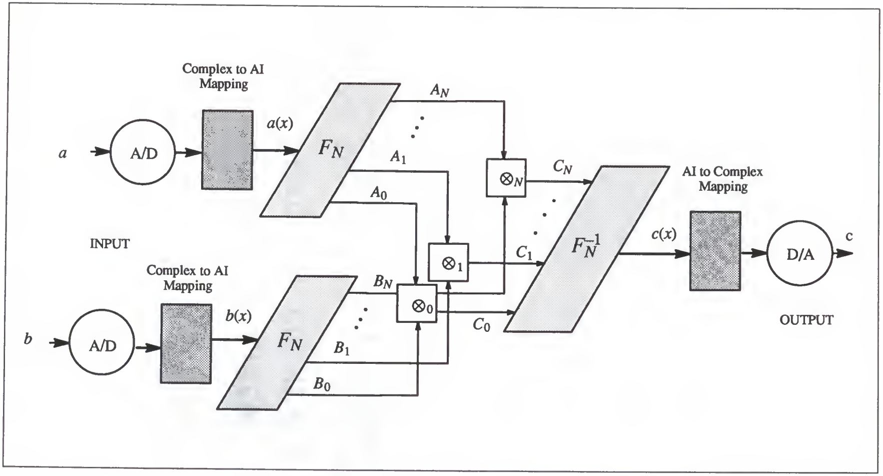


Figure 4.2 The Pictorial Representation of the *N*th-order Single Modulus PRNS System

and addition and multiplication in the extension field are defined by

$$[\,(a+\hat{j}\,b)+(c+\hat{j}\,d)\,] \bmod M=[\,(a+c)+\hat{j}\,(b+d)\,] \bmod M \tag{4.40}$$

$$[(a+\hat{j}\,b)\cdot(c+\hat{j}\,d)] \bmod M=[\,(ac-bd)+\hat{j}\,(ab+cd)\,] \bmod M. \tag{4.41}$$

It is known [Har65] that for all primes of the form $M = 4k + 3$, -1 is a quadratic nonresidue mod M, where k is an integer. This defines a set of primes that can be selected for forming extension fields where the quadratic congruence is solvable. For example, let $k = 1$, then the prime $M = 7$ causes the extension field GF(7^2) of 49 elements to have solution to the quadratic equation; likewise, for $k = 4$, there exists the extension field GF(19^2) of 361 elements.

1) Prime finite field index calculus CRNS

Every prime field GF(M) has an associated index set that is analogous to a set of logarithms in the real number system [see Chapter 2]. By using this index set and adding the indexes mod M–1, finite field multiplication can be implemented with a mod M–1 adder and a set of index tables stored in high-speed memory which is of the order of (log M). For example, the four real multiplications required for the complex multiply of Equation (4.41) can be replaced by four real additions and a nominal amount of memory storage.

2) Complex extension field index calculus CRNS

The complex extension field GF(M^2) also has a real index set that is generated by a complex primitive element of order M^2–1. The existence of a real index calculus implies that complex multiplication in GF(M^2) can be replaced by a single real addition and a table of indexes. Since GF(M^2) contains M^2–1 nonzero elements, the index addition is mod M^2–1 addition, and the table of indexes will be of the order of (log M^2).

<u>ICPRNS.</u> Similar to the QRNS, for certain primes, the quadratic equation $x^N \equiv -1$ mod M does not have a solution in GF(M). According to previous chapters, a solution to the congruence (or equation) can be found in the Nth order extension field GF(M^N) (or $\mathbf{Z}_{M^N}$) and is denoted $\hat{r}$. Under these circumstances the set of elements in GF(M^N) is

$$\mathbf{Z}_{M^N} = \mathbf{Z}_M(\hat{r}) = \{z_0 + z_1\hat{r}^1 + \ldots + z_{N-1}\hat{r}^{N-1} : z_i \in \mathbf{Z}_M\}, \tag{4.42}$$

The rules of composition of two elements u and v of $\mathbf{Z}_{M^N}$ when added yield $u + v = [\, u_0 + v_0,\, u_1 + v_1, \ldots, u_N + v_N\,]$. With multiplication defined by

$$uv = c = \sum_{j=0}^{N} c_j r^j \tag{4.43}$$

where

$$c_k = \sum_{j=0}^{k} a_{k-j} b_j - \sum_{j=k+1}^{N} a_j b_{N-j+k}\ .$$

For all primes not having the form $M = 2Nk + 1$, this defines a set of primes that can form extension fields where the congruence is solvable. For example, let $N = 4$, then the prime $M = 11$ results in the extension field GF(11^4) of 14,641 elements.

The algebraic integer extension field GF(M^N) also has a real index set that is generated by a complex primitive element of order $M^N - 1$. The existence of a real index calculus implies that complex multiplication in GF(M^N) can be replaced by a single real addition and a table of indexes. Since GF(M^N) contains $M^N - 1$ nonzero elements, the index addition is mod $M^N - 1$ addition, and the table of indexes will be of the order of ($\log M^N$).

4.4.2 Finite Extension Field Transforms

Transforms Over Extension Field GF(p^m). Transforms analogous to the discrete Fourier transform can be defined in finite fields and also be calculated efficiently by the FFT algorithms. For the finite field GF(p^m), let d be a divisor of $p^m - 1$ (possibly $d = p^m - 1$), and α be an element of order d in the multiplicative group G^* of GF(p^m) which is $\{1, \alpha, \alpha^2, \ldots, \alpha^{d-1}\}$. Then one can define the transform $T[a]$ of a sequence $a = \{ a_i : i = 0, 1, 2, \ldots, d-1 \}$ of elements of GF(p^m) to be the sequence $A = \{ A_k : k = 0, 1, 2, \ldots, d-1 \}$ where

$$A_k = \sum_{i=0}^{d-1} a_i \, \alpha^{ik} . \tag{4.44}$$

Its inverse transform is

$$a_i = D \cdot \sum_{k=0}^{d-1} A_k \, \alpha^{-ik} \tag{4.45}$$

where D is the integer for which

$$D \cdot d \equiv 1 \bmod p^m - 1 . \tag{4.46}$$

The transform pair in Equations (4.44) and (4.45) may be calculated by FFT algorithms, which is simplest to perform when the integer d is highly composite. The total number of operations is reduced to $O(d \log d)$ from $O(d^2)$ operations required to calculate these transforms in the most obvious way. The principal reason for interest in the transform (4.44) lies in the following "Cyclic Convolution Property" (CCP). Suppose that three pairs of sequences a, A, and b, B, and c, C all of length d form transform pairs as those of Equations (4.44) and (4.45) that

$$C_i = A_i \cdot B_i , \quad 0 \le i \le d-1 , \tag{4.47}$$

then

$$c_i = \sum_{j=0}^{d-1} \sum_{k=0}^{d-1} a_i \, b_k , \tag{4.48}$$

where $j + k \equiv i \bmod d$ and $0 \le i \le d-1$. If we extend the definition of b to all i by making the sequence period with period d, Equation (4.48) may be written

$$c_i = \sum_{j=0}^{d-1} a_j b_{(i-j) \bmod d} . \tag{4.49}$$

Thus, the calculation of the "cyclic convolution" of the sequences a and b, as defined by Equation (4.48), may be obtained by transforming the sequences, multiplying the results term-by-term as in Equation (4.47), and performing the inverse transform (4.45).

To show the CCP of Equations (4.47) and (4.48), we derive the discrete delta function $\delta_d(l)$, observe first that all element x of G^* satisfy the equation

$$x^d - 1 = 0 . \tag{4.50}$$

However since the equation factors as

$$x^d - 1 = (x - 1)\sum_{k=0}^{d-1} x^k , \tag{4.51}$$

for $x \neq 1$, one has

$$\sum_{k=0}^{d-1} x^k = 0 . \tag{4.52}$$

Now consider the sum of α^l , where $\alpha \in G^*$ and $\alpha = x^k$, this is

$$\sum_{k=0}^{d-1} (x^k)^l = \sum_{k=0}^{d-1} (x^l)^k , \tag{4.53}$$

For the case of $l \equiv 0 \bmod d, \beta \in G^*$ and $\beta = x^l$, from Equation (4.52) it becomes

$$\sum_{k=0}^{d-1} \beta^k = \sum_{k=0}^{d-1} x^{lk} = 0 , \tag{4.54}$$

however for $l \equiv 0 \bmod d$, $x^l = 1$,

$$\sum_{k=0}^{d-1} x^{lk} = d . \tag{4.55}$$

From the last two equations, we represent

$$\sum_{k=0}^{d-1} x^{lk} = d \cdot \delta_d(l) .$$

By Equation (4.45) the inverse transform of C_k is

$$c_i = D \cdot \sum_{k=0}^{d-1} C_k \alpha^{-ik}$$

$$= D \cdot \sum_{k=0}^{d-1} (A_k \cdot B_k)\, \alpha^{-ik}$$

$$= D \cdot \sum_{k=0}^{d-1} (\sum_{j=0}^{d-1} a_j \alpha^{jk})(\sum_{m=0}^{d-1} b_m \alpha^{mk}) \alpha^{-ki}$$

$$= D \cdot \sum_{j=0}^{d-1} \sum_{m=0}^{d-1} a_j b_m (\sum_{k=0}^{d-1} \alpha^{k(j+m-i)})$$

$$= D \cdot \sum_{j=0}^{d-1} \sum_{m=0}^{d-1} a_j b_m (d \cdot \delta_d(j+m-i))$$

$$= D \cdot d \sum_{j=0}^{d-1} \sum_{m=0}^{d-1} a_j b_m \, .$$

With $j + m \equiv i \bmod d$ we conclude that

$$c_i = \sum_{j=0}^{d-1} a_j b_{(i-j) \bmod d} \, .$$

Conjugacy Property of Finite Field. Elements of the field with the same minimal polynomial are called conjugates with respect to GF(p). For $\beta \in$ GF(p^m), the p powers of β fall into disjoint conjugate sets. A typical conjugate set B is shown as

$$B = \{ \beta, \beta^p, \beta^{p^2}, \ldots, \beta^{p^{l-1}} \} \tag{4.56}$$

where l is called the coset length and represents the smallest positive integer such that $\beta^{p^l} \equiv \beta$. The conjugate sets can also be defined in terms of the exponents of β where the operation of multiplying the exponents by p divides the integers modulo ($p^m - 1$) into sets of conjugate class called the cyclotomic cosets. A cyclotomic coset C containing s consists of the set

$$C = \{ s, sp, sp^2, \ldots, sp^{l_s-1} \}, \tag{4.57}$$

where l_s is the smallest positive integer that satisfies

$$p^{l_s} s \equiv s \bmod p^m - 1 \, . \tag{4.58}$$

The coset is then denoted by C_s, where s called the coset leader is the smallest number in the coset. Some properties of the coset length are (1) if $0 \in C_s$, then $l_s = 1$; (2) given any divisor t of m (excluding $t = 1$ when $p = 2$), $\theta = (p^m - 1)/(p^t - 1) \in C_s$, $l_s = t$; and (3) With θ as in (2) and α a primitive element of GF(2^m), α^θ generates the subfield GF(p^{l_s}) of GF(p^m) and consequently $l_s \mid n$. Note that for $s = 0$ it is considered as a coset of length 1.

Example 4.4 The cyclotomic cosets of small finite fields.

Consider the finite field GF(2^4). Let $d = 2^4 - 1$, then d_i is { 1, 3, 5, 15 }. Because 3 divides $2^2 - 1$ and ϕ (3) = 2, there is one coset of length 2 with elements of order 3. Since 15 divides $2^4 - 1$, there are 2 (= $\phi(15)/4$) cosets of length 4 with elements of order 15. Lastly 5 divides $2^4 - 1$, there exists a coset of length 4 with elements of order 5. A summary is listed below.

C_0 = { 0 } – element of order 1

C_1 = { 1, 2, 4, 8 } – elements of order 15

C_3 = { 3, 6, 12, 9 } – elements of order 5

C_5 = { 5, 10 } – elements of order 3

C_7 = { 7, 14, 13, 11 } – elements of order 15.

For the finite field GF(2^5), follow the procedures in the previous case. One finds 6 (= $\phi(31)/5$) cosets of length 5 with elements of order 31 and one coset of length 1. These cosets are

C_0 = { 0 }

C_1 = { 1, 2, 4, 8, 16 }

C_3 = { 3, 6, 12, 24, 17 }

C_5 = { 5, 10, 20, 9, 18 }

$C_7 = \{ 7, 14, 28, 25, 19 \}$

$C_{11} = \{ 11, 22, 13, 26, 21 \}$

$C_{15} = \{ 15, 30, 29, 27, 23 \}$. ♦

Given an integer d with $d \mid (p^m - 1)$, the number of cyclotomic cosets, denoted by c, and the length of a coset, l, can be calculated as follows. Let d_i be a divisor of d for $i = 1, 2, \ldots, t$. Note that Euler's theorem indicates d_i is also the possible order of the elements in the finite field GF(p^m). There exists c_i cyclotomic cosets of length l_i where $c_i = \phi(d_i)/l_i$, l_i is the least positive integer such that $p^{l_i} \equiv 1 \bmod d_i$. Hence, the number of the cyclotomic coset, c, of the field of order d becomes

$$c = \sum_{d_i \mid p^{l_i} - 1} c_i \ . \tag{4.59}$$

<u>Example 4.5</u> The number of cyclotomic coset.

1) In the case of the finite field, GF(2^8), since $d \mid (2^8 - 1)$, let $d = 85$. Hence, $d_i = \{ 85, 17, 5, 1 \}$. For $d_0 = 85$, $c_0 = \phi\,(d_0)/l_0 = \phi\,(85)/l_0 = 64/l_0$. Since $l_0 = 8$ is the least positive integer such that $2^8 \equiv 1 \bmod d_0$, $c_0 = 8$ which interprets that there are 8 cosets of length 8. Similarly, for $d_1 = 17$, $c_1 = \phi\,(17)/l_1 = 16/l_1$ and $17 \mid (2^8 - 1)$ which indicates $l_1 = 8$, therefore, $c_1 = 2$. For $d_2 = 5$, $c_2 = \phi\,(5)/4 = 1$. For $d_3 = 1$, $c_3 = 1$. Thus, total number of the cosets in the case of $d = 85$ is 12.

2) Let $d = 17$, then $d_i = \{ 17, 1 \}$. For $d_0 = 17$, $c_0 = \phi\,(d_0)/l_0 = \phi\,(17)/l_0 = 16/l_0$. Since $l_0 = 8$ is the least positive integer such that $2^8 \equiv 1 \bmod d_0$, $c_0 = 2$ which interprets that there are 2 cosets of length 8. For $d_1 = 1$, $c_1 = 1$. Thus, total number of the cosets in the case of $d = 17$ is 3.

♦

In previous chapter we note that p^i th powers of a field element in the finite field GF(p^m) of characteristic p falls into the same cyclotomic coset. This conjugacy property of the cyclotomic cosets in the finite field GF(p^m) can be applied to expedite finite field transforms.

Theorem 4.1: For V a vector of length d of elements of GF(p^m), where d is a divisor of p^m-1, then the inverse Fourier transform v is a vector of elements of GF(p) if and only if the following equations are satisfied

$$V_j^p = V_{(pj) \bmod d}, \tag{4.60}$$

where $V_j \in V$ and $j = 0, 1, \cdots, d-1$.

Proof. By definition of the finite field transformation in Equation (4.44) and the power forming property in Corollary of Equation (2.31), it becomes

$$V_j^p = \left(\sum_{i=0}^{d-1} v_i \alpha^{ij}\right)^p$$

$$= \sum_{i=0}^{d-1} v_i^p \alpha^{pij} .$$

Further if v_i is an element of GF(p) for all i, then $v_i^p = v_i$. Consequently this gives

$$V_j^p = \sum_{i=0}^{d-1} v_i \alpha^{i(pj)}$$

$$= V_{(pj) \bmod d} .$$

Conversely, suppose that for all j, $V_j^p = V_{(pj) \bmod d}$. Then

$$\sum_{i=0}^{d-1} v_i^p \alpha^{pij} = \sum_{i=0}^{d-1} v_i \alpha^{i(pj)},$$

where $j = 0, 1, \cdots, d-1$. Let $k = pj$. Because p is relatively prime to d, as j ranges over all values between 0 and $d-1$, k also takes on all values between 0 and $d-1$. Hence

$$\sum_{i=0}^{d-1} v_i^p \alpha^{ik} = \sum_{i=0}^{d-1} v_i \alpha^{ik}, \tag{4.61}$$

where $k = 0, 1, \cdots, d-1$. And by uniqueness of the Fourier transform, $v_i^p = v_i$ for all i. Thus v_i is a zero of $x^p - x$ for all i, and such zeros are all elements of GF(p). ♦

This theorem asserts that if the time-domain signal v is in GF(p), then the value of the spectrum V_j in the cyclotomic coset C specifies the value of the spectrum at all other frequencies (or elements) in C.

Normal Basis Applications. For $V \in$ GF(p^m), $N_n = \{ \alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{m-1}} \}$ a normal basis. Thus,

$$V = v_0\alpha + v_1\alpha^p + v_2\alpha^{p^2} + \ldots + v_{m-1}\alpha^{p^{m-1}}$$

where $v_i \in$ GF(p), i = 0, 1, . . . , $m-1$. According to Theorem 2.3.

$$V^p = v_0^p\alpha^p + v_1^p\alpha^{p^2} + v_2^p\alpha^{p^3} + \ldots + v_{m-1}^p\alpha^{p^m}$$

$$= v_{m-1}\alpha + v_0\alpha^p + v_1\alpha^{p^2} + v_2\alpha^{p^3} + \ldots + v_{m-2}\alpha^{p^{m-1}}.$$

Cyclic Shift Property. From the normal basis property shown above, the operation of taking pth powers of V is equivalent to continuing cyclic shifts. A pictorial presentation of the evaluation of V^{p^i} is shown in Figure 4.3.

Example 4.6 Consider GF(2^4) $\cong Z_2[x]/(x^4 + x^3 + 1)$. Two of normal bases are illustrated, $N_{n1} = \{ \alpha, \alpha^2, \alpha^4, \alpha^8 \} = \{ x, x^2, x^3 + 1, x^3 + x^2 + x \}$, which is called primitive normal basis due to the fact that x is a primitive element of the field, and the other one $N_{n2} = \{ \alpha^3, \alpha^6, \alpha^{12}, \alpha^9 \} = \{ x^3, x^3 + x^2 + x + 1, x + 1, x^2 + 1 \}$. Field elements are represented under various

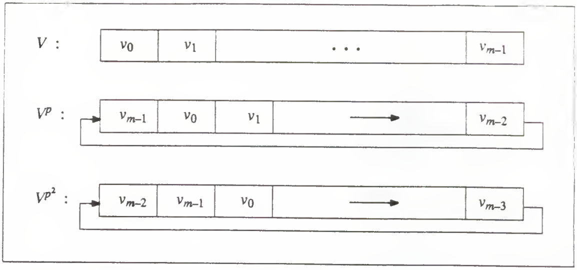


Figure 4.3 The Cyclic Shift Property of Normal Basis Representation of V

bases and shown as following table. To show the cyclic shift property, let $V = \alpha^6$, then $V^2 = \alpha^{12}$, $V^4 = \alpha^{24} = \alpha^9$, and $V^8 = \alpha^{48} = \alpha^3$. From Table 4.3, We look for the primitive normal basis representation of V, V^2, V^4, V^8 which are (1, 1, 1, 0), (0, 1, 1, 1), (1, 0, 1, 1), and (1, 1, 0, 1), respectively. Similarly, the normal basis representation of V, V^2, V^4, V^8 are (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), and (1, 0, 0, 0), respective. These shows the property. For both normal bases, the element 1 has basis representation (1, 1, 1, 1). ♦

Table 4.3 Field Elements of GF(2^4) Represented Under Primal, Primitive Normal, and Normal Bases

Elements	Primal basis (a_0, a_1, a_2, a_3)	Primitive Normal Basis (p_0, p_1, p_2, p_3)	Normal Basis (n_0, n_1, n_2, n_3)
$\alpha^{-\infty} = 0$	(0, 0, 0, 0)	(0, 0, 0, 0)	(0, 0, 0, 0)
$\alpha^0 = 1$	(1, 0, 0, 0)	(1, 1, 1, 1)	(1, 1, 1, 1)
$\alpha = x$	(0, 1, 0, 0)	(1, 0, 0, 0)	(1, 1, 0, 1)
$\alpha^2 = x^2$	(0, 0, 1, 0)	(0, 1, 0, 0)	(1, 1, 1, 0)
$\alpha^3 = x^3$	(0, 0, 0 ,1)	(1, 1, 0, 1)	(1, 0, 0, 0)
$\alpha^4 = x^3 + 1$	(1, 0, 0, 1)	(0, 0, 1, 0)	(0, 1, 1, 1)
$\alpha^5 = x^3 + x + 1$	(1, 1, 0, 1)	(1, 0, 1, 0)	(1, 0, 1, 0)
$\alpha^6 = x^3 + x^2 + x + 1$	(1, 1, 1, 1)	(1, 1, 1, 0)	(0, 1, 0, 0)
$\alpha^7 = x^2 + x + 1$	(1, 1, 1, 0)	(0, 0, 1, 1)	(1, 1, 0, 0)
$\alpha^8 = x^3 + x^2 + x$	(0, 1, 1, 1)	(0, 0, 0, 1)	(1, 0, 1, 1)
$\alpha^9 = x^2 + 1$	(1, 0, 1, 0)	(1, 0, 1, 1)	(0, 0, 0, 1)
$\alpha^{10} = x^3 + x$	(0, 1, 0, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)
$\alpha^{11} = x^3 + x^2 + 1$	(1, 0, 1, 1)	(0, 1, 1, 0)	(1, 0, 0, 1)
$\alpha^{12} = x + 1$	(1, 1, 0, 0)	(0, 1, 1, 1)	(0, 0, 1, 0)
$\alpha^{13} = x^2 + x$	(0, 1, 1, 0)	(1, 1, 0, 0)	(0, 0, 1, 1)
$\alpha^{14} = x^3 + x^2$	(0, 0, 1, 1)	(1, 0, 0, 1)	(0, 1, 1, 0)

CHAPTER 5
FAST DISCRETE FOURIER TRANSFORMS OVER FINITE FIELDS

5.1 Introduction

The transform with the cyclic convolution property (CCP) over the extension field GF(p^m) is shown to be an effective application for digital signal processing. In this chapter the Good-Thomas prime-factor FFT algorithm is applied over the Galois field GF(p^m). The intermediate results are represented by using a normal basis representation. A significant computational reduction is obtained by applying a conjugacy relation to a cyclotomic coset of the intermediate variables, and by using the cyclic-shift property of p powers of the variables within the normal basis representation. Once the VLSI architecture for the butterfly module of a cyclotomic coset based on the algorithm is developed, these module arrays are used to form the stages of the fast transform. For the case of $p = 2, m = 8$, performance analysis shows a six-fold reduction in computational complexity which is achieved in terms of an added hardware budget.

5.2 Fast Prime-Factor Finite Field Transform

Blahut [Bla83] defined the discrete Fourier transforms over the Galois field GF(p^m). We consider $\boldsymbol{v} = \{ \ v_i \ : i = 0, 1, \ldots, d-1 \ \}$ a vector over GF(p), where d divides $p^m - 1$ for some m, and r an element of GF(p^m) of order d. The Galois field transform of the vector $\boldsymbol{v}$ is $\boldsymbol{V} = \{ \ V_j \ : j = 0, 1, \ldots, d-1 \ \}$, where $V_j \in$ GF(p^m), The transform pair is given by

$$V_j = \sum_{i=0}^{d-1} r^{ij} \, v_i , \tag{5.1}$$

and

$$v_i = d^{-1} \sum_{j=0}^{d-1} r^{-ij} V_i \ . \tag{5.2}$$

where d^{-1} denotes the inverse of d and satisfies $d^{-1} d \equiv 1 \bmod p$.

This transform can be expedited by traditional FFT-type algorithms, but still has the computational complexity of O($d\log d$). This research introduces the concept of normal basis representation in GF(p^m) and the conjugacy property of finite field elements. Application of these concepts to the existing prime-factor fast transform not only serves to simplify the transform, but also leads to the development of a very compact device. This paper demonstrates the forward transform. The inverse transform can also be processed based upon this concept.

The Good-Thomas prime-factor FFT algorithm is used to integrate a set of DFTs over a Galois field [Red86]. This algorithm relys upon index expansions based on the Chinese remainder theorem (CRT). The development of the normal basis Good-Thomas FFT over a Galois field is summarized as follows. Consider the case

$$d = \prod_{i=1}^{s} p_i \ , \ (\ p_i, \ p_j \ ) \ = \ 1, \ \ i \neq j, \text{ and let}$$

$$d_k = \prod_{i=k}^{s} p_i \ , 1 \ < \ k \ \leq s \, .$$

There are s stages in the fast transform which the initial stages based on p_s and the last stage based on p_1; intermediate results are represented by the symbol X.

At the first stage the scrambled input and output indices i and j are expanded

$$i \ = \ i_1(d_2 D_2) \ + \ i_2(p_1 C_1) \, , \tag{5.3}$$

$$j \ = \ j_1 \ d_2 + j_2 \ p_1 \, , \tag{5.4}$$

where $i_1 \equiv i \bmod p_1$, $i_2 \equiv i \bmod d_2$, $j_1 \equiv jD_2 \bmod p_1$, and $j_2 \equiv jC_1 \bmod d_2$. Note that C_1 and D_2 are obtained by virtue of the fact that p_1 and d_2 are relatively prime and by using the Euclidean algorithm such that

$$1 = p_1C_1 + d_2D_2. \tag{5.5}$$

The transform may be expressed in the following form

$$\begin{aligned}
V_j &= \sum_{i=0}^{d-1} r^{ij} \; v_i \\
&= \sum_{i_2=0}^{d_2-1} \sum_{i_1=0}^{p_1-1} r^{(i_1d_2D_2+i_2p_1C_1)(j_1d_2+j_2p_1)} \;\; v_i \\
&= \sum_{i_2=0}^{d_2-1} (r^{p_1})^{i_2j_2p_1C_1} \; [\sum_{i_1=0}^{p_1-1} (r^{d_2})^{i_1j_1(d_2D_2)} \; v_i \;] \\
&= \sum_{i_2=0}^{d_2-1} (r^{p_1})^{i_2j_2p_1C_1} \; X_{j_1,i_2} \; .
\end{aligned}$$

By applying the relation in Equation (5.5) to the above equation, we have

$$\begin{aligned}
(r^{p_1})^{i_2j_2p_1C_1} &= (r^{p_1(1-d_2D_2)})^{i_2j_2} \\
&= (r^{p_1}r^{-d_2D_2})^{i_2j_2} \\
&= (r^{p_1})^{i_2j_2} \, .
\end{aligned} \tag{5.6}$$

Thus, the transform becomes

$$V_j = \sum_{i_2=0}^{d_2-1} (r^{p_1})^{i_2j_2} \; X_{j_1,i_2} \, . \tag{5.7}$$

The next stage expands the indices i_2 and j_2 by a similar procedure as those in Equations (5.3) and (5.4)

$$\begin{aligned}
i_2 &= i_2'(d_3D_3) + i_3(p_2C_2), \\
j_2 &= j_2' \, d_3 + j_3 \;\; p_2 \, ,
\end{aligned} \tag{5.8}$$

where $i_2' \equiv i \bmod p_2$, $i_3 \equiv i \bmod d_3$, and

$$\begin{aligned} j_2' &\equiv j_2 D_3 \bmod p_2 \\ &= ((jC_1 \bmod d_2)\, D_3) \bmod p_2 \\ &= ((jC_1 \bmod d_2) \bmod p_2)\ (D_3 \bmod p_2) \\ &= (jC_1 \bmod p_2)\ (D_3 \bmod p_2) \\ &= jC_1 D_3 \bmod p_2\,, \end{aligned} \tag{5.9}$$

and

$$\begin{aligned} j_3 &\equiv j_2 C_2 \bmod d_3 \\ &= ((jC_1 \bmod d_2) C_2) \bmod d_3 \\ &= jC_1 C_2 \bmod d_3\,. \end{aligned} \tag{5.10}$$

Similar to the initial stage, p_2 and d_3 are relatively prime such that

$$1 = p_2 C_2 + d_3 D_3\,. \tag{5.11}$$

The output of this stage is given by employing X_{j_1,i_2} of the output in the first stage

$$\begin{aligned} V_j &= \sum_{i_2=0}^{d_2-1} (r^{p_1})^{i_2 j_2}\, X_{j_1,i_2} \\ &= \sum_{i_3=0}^{d_3-1} \sum_{i_2'=0}^{p_2-1} (r^{p_1})^{(i_2' d_3 D_3 + i_3 p_2 C_2)(j_2' d_3 + j_3 p_2)}\, X_{j_1,i_2} \\ &= \sum_{i_3=0}^{d_3-1} (r^{p_1 p_2})^{i_3 j_3 p_2 C_2} \left[\ \sum_{i_2'=0}^{p_2-1} (r^{p_1 d_3})^{i_2' j_2' (d_3 D_3)}\, X_{j_1,i_2}\ \right] \\ &= \sum_{i_3=0}^{d_3-1} (r^{p_1 p_2})^{i_3 j_3 p_2 C_2}\, X_{j_1,j_2',i_3}\,. \end{aligned}$$

Similar to Equation (5.6), we apply Equation (5.11) and yield

$$\begin{aligned}(r^{p_1p_2})^{i_3j_3p_2C_2} &= (r^{p_1p_2(1-d_3D_3)})^{i_3j_3} \\ &= (r^{p_1p_2}r^{-d_3D_3})^{i_3j_3} \\ &= (r^{p_1p_2})^{i_3j_3} .\end{aligned}$$

Thus, the transform to this point becomes

$$V_j = \sum_{i_3=0}^{d_3-1} (r^{p_1p_2})^{i_3j_3} X_{j_1j_2',i_3} . \tag{5.12}$$

The general stage in the Good-Thomas algorithm is expressible in terms of the following equations. At stage k, the scrambled input and output indices becomes

$$\begin{aligned} i_k &= i_k'(d_{k+1}D_{k+1}) + i_{k+1}(p_kC_k) , \qquad (5.13) \\ j_k &= j_k'\, d_{k+1} + j_{k+1}\, p_k ,\end{aligned}$$

with

$$\begin{aligned} j_k' &\equiv [\, j(C_1C_2 \,.\,.\,.\, C_{k-1})D_{k+1} \,] \bmod p_k , \\ j_{k+1} &\equiv j(C_1C_2 \,.\,.\,.\, C_k) \bmod d_{k+1}, \end{aligned}$$

where C_k and D_{k+1} satisfy the equation

$$1 = p_kC_k + d_{k+1}D_{k+1} . \tag{5.14}$$

The output of this stage is analogous to the procedures shown earlier.

$$\begin{aligned} V_j &= \sum_{i_k=0}^{d_k-1} (r^{p_1p_2 \cdots p_{k-1}})^{i_kj_k} X_{j_1j_2',\ldots,j_{k-1}',i_k} , \\ &= \sum_{i_{k+1}=0}^{d_{k+1}-1} \sum_{i_k'=0}^{p_k-1} (r^{d/d_{k+1}})^{(i_k'd_{k+1}D_{k+1}+i_{k+1}p_kC_k)(j_k'd_{k+1}+j_{k+1}p_k)} X_{j_1j_2',\ldots,j_{k-1}',i_k} \\ &= \sum_{i_{k+1}=0}^{d_{k+1}-1} (r^{d/d_{k+1}})^{i_{k+1}j_{k+1}} \Big[\sum_{i_k'=0}^{p_k-1} (r^{d/p_k})^{i_k'j_k'(d_{k+1}D_{k+1})} X_{j_1j_2',\ldots,j_{k-1}',i_k} \Big] \\ &= \sum_{i_{k+1}=0}^{d_{k+1}-1} (r^{d/d_{k+1}})^{i_{k+1}j_{k+1}} X_{j_1j_2',\ldots,j_k',i_{k+1}} , \qquad (5.15)\end{aligned}$$

where the output index j expanded to stage k is defined as

$$\begin{aligned} j &= j_1 d_2 + j_2' p_1 d_3 + j_3' p_1 p_2 d_4 + \ . \ . \ . + j_k' p_1 p_2 \ldots p_{k-1} d_{k+1} + j_{k+1} p_1 p_2 \ldots p_k \\ &= [j_1(d/p_1) + j_2'(d/p_2) + \ . \ . \ . + j_k'(d/p_k) + j_{k+1}(d/d_{k+1})] \bmod d. \end{aligned} \tag{5.16}$$

Since the intermediate results of the Good-Thomas transform are the elements of the finite field GF(p^m), the conjugacy property is applicable and the transform process is simplified. Letting $t = 1, 2, \ldots, l$, the notation $R_{i,t}$ is defined as

$$R_{i,t} \equiv j_i p^t \bmod p_i \quad \text{, for } i = 1, 2, \ldots, k. \tag{5.17}$$

If the output of the initial stage in Equation (5.7) is raised it to the pth power

$$\begin{aligned} (X_{j_1,i_2})^{p^t} &= \Big(\sum_{i_1=0}^{p_1-1} (r^{d_2})^{i_1 j_1 (d_2 D_2)} \ v_i \Big)^{p^t} \\ &= \sum_{i_1=0}^{p_1-1} (r^{d_2})^{i_1 j_1 p^t (d_2 D_2)} \ v_i^{p^t} , \end{aligned} \tag{5.18}$$

then according to the finite field properties discussed in Chapter 3 and using Equation (5.17) leads to the following general expression of the result of the initial stage

$$\begin{aligned} (X_{j_1,i_2})^{p^t} &= \sum_{i_1=0}^{p_1-1} (r^{d_2})^{i_1 R_{1,t} (d_2 D_2)} \ v_i \\ &= X_{R_{1,t},i_2} . \end{aligned}$$

Similarly, the output of the second stage is obtained and shown as follows.

$$\begin{aligned} (X_{j_1,j_2',i_3})^{p^t} &= \sum_{i_2'=0}^{p_2-1} (r^{d/p_2})^{i_2' j_2' p^t (d_3 D_3)} \ (X_{j_1,i_2})^{p^t} \\ &= \sum_{i_2'=0}^{p_2-1} (r^{d/p_2})^{i_2' R_{2,t} (d_3 D_3)} \ X_{R_{1,t},i_2} \\ &= X_{R_{1,t},R_{2,t},i_3} . \end{aligned} \tag{5.19}$$

The general form of the conjugacy property is demonstrated by raising the *p*th power of the defining items $X_{j_1,j'_2,\ldots,j'_k,i_{k+1}}$ in Equation (5.15) for the intermediate results from stage *k*

$$
\begin{aligned}
(X_{j_1,j'_2,\ldots,j'_k,i_{k+1}})^{p'} &= \sum_{i'_k=0}^{p_k-1} (r^{d/p_k})^{i'_k j'_k p'(d_{k+1}D_{k+1})}\, X^{p'}_{j_1,j'_2,\ldots,j'_{k-1},i_k} \\
&= \sum_{i'_k=0}^{p_k-1} (r^{d/p_k})^{i'_k R'_{k,s}(d_{k+1}D_{k+1})}\, X_{R_{1,s},R'_{2,s},\ldots,R'_{k-1,s},i_k} \\
&= X_{R_{1,s},R'_{2,s},\ldots,R'_{k,s},i_{k+1}}\,. \qquad (5.20)
\end{aligned}
$$

These intermediate results are carried through to the final output values where output V_j is expressed by expanding *j* according to Equation (5.16)

$$
\begin{aligned}
V_j^{p'} &= \sum_{i_s=0}^{p_s-1} (r^{d/d_s})^{i_s j_s p'}\, (X_{j_1,j'_2,\ldots,j'_{s-1},i_s})^{p'} \\
&= V_{[R_{1,s}(d/p_1)+R'_{2,s}(d/p_2)+\ldots+R'_{s,s}(d/p_s)] \bmod d}\,. \qquad (5.21)
\end{aligned}
$$

5.2.1 <u>The Application of Normal Basis and Conjugacy Properties</u>

MacWilliams and Sloan [Mac77] (also see Chapter 3 and 4) suggest that there always exists a normal basis in the finite field GF(p^m) for all positive integers m. In other words, one can find a field element α such that $N_n = \{\alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{m-1}}\}$ is a basis set of GF(p^m). Thus, the field element $V_j \in$ GF(p^m) can be uniquely expressed in terms of the basis N_n as

$$V_j = v_{j,0}\alpha + v_{j,1}\alpha^p + v_{j,2}\alpha^{p^2} + \ldots + v_{j,m-1}\alpha^{p^{m-1}}, \qquad (5.22)$$

where $v_{j,i} \in$ GF(p), $i = 0, 1, \ldots, m-1$. According to the binomial theorem and the properties of the finite field GF(p^m), the *p*th power of V_j becomes

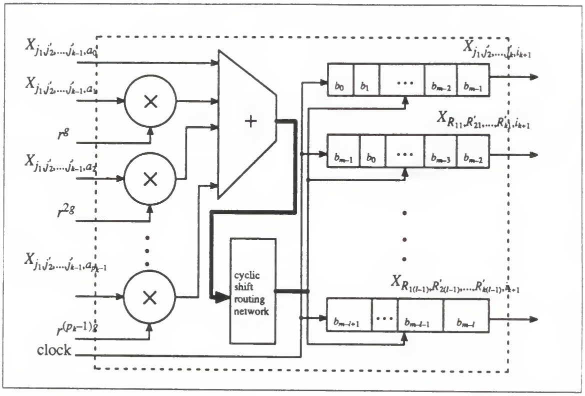
$$V_j^p = v_{j,m-1}\alpha + v_{j,0}\alpha^p + v_{j,1}\alpha^{p^2} + \ldots + v_{j,m-2}\alpha^{p^{m-1}}. \qquad (5.23)$$

Hence, V_j^p is simply a cyclic shift of V_j in the normal basis representation.

As stated in Chapter 4, elements of the field GF(p^m) with the same minimal polynomial are called conjugates with respect to GF(p). For $\alpha \in$ GF(p^m), the p powers of α fall into disjoint conjugate sets.

5.2.2 A Normal Basis Architecture for the Finite Field Transform

A modularized VLSI circuitry is developed for the butterfly module of a cyclotomic coset in Figure 5.1. The module consists of p_k-1 normal basis multipliers, a p_k-input digit-



$X_{j_1 j'_2 \ldots j'_{k-1}, a_0}$
$X_{j_1 j'_2 \ldots j'_{k-1}, a_1}$
r^g
$X_{j_1 j'_2 \ldots j'_{k-1}, a_2}$
r^{2g}
$X_{j_1 j'_2 \ldots j'_{k-1}, a_{p_k-1}}$
$r^{(p_k-1)g}$
clock
+
cyclic shift routing network
$X_{j_1 j'_2 \ldots j'_k, i_{k+1}}$
b_0 b_1 … b_{m-2} b_{m-1}
$X_{R_{11}, R'_{21}, \ldots, R'_{k1}, i_{k+1}}$
b_{m-1} b_0 … b_{m-3} b_{m-2}
$X_{R_{1(l-1)}, R'_{2(l-1)}, \ldots, R'_{k(l-1)}, i_{k+1}}$
b_{m-l+1} … b_{m-l-1} b_{m-l}

Figure 5.1 The VLSI Butterfly Module CB of a Cyclotomic Coset of Length l

wise adder, a cyclic shift routing network, and l_r m-digit registers. Generation of the l intermediate variables occurs in the coset module where there is only one variable which needs to be evaluated. The operations include p_k-1 multiplications and additions. The remaining l – 1 variables are obtained by a cyclic-shifting of the evaluated variable within a pipeline clock period. Note that in Figure 5.1

$$r^{i''_k g} = (r^{n/p_k})^{i''_k j''_k (n_{k+1} D_{k+1})} , \tag{5.24}$$

where $i'_k = 0, 1, \ldots, p_k - 1, k = 1, 2, \ldots, s-1$.

These modules serve as basic elements in the hierarchical design of the Galois-field transform system shown in Figure 5.2 where the transform is based on p_s in stage one, on

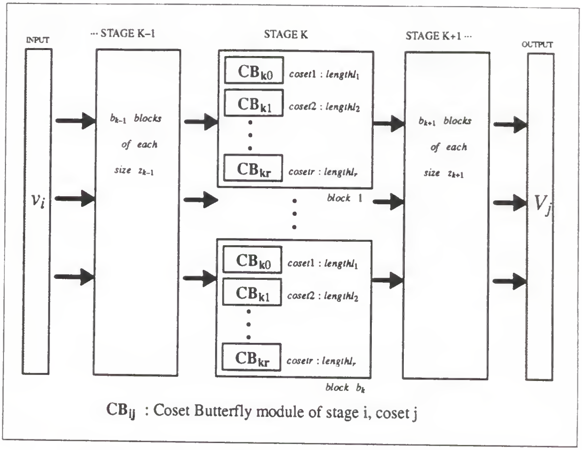

INPUT
··· STAGE K–1
STAGE K
STAGE K+1 ···
OUTPUT
V_i
b_{k-1} blocks of each size z_{k-1}
CB_{k0} coset1 : $lengthl_1$
CB_{k1} coset2 : $lengthl_2$
CB_{kr} cosetr : $lengthl_r$
block 1
block b_k
b_{k+1} blocks of each size z_{k+1}
V_j
CB_{ij} : Coset Butterfly module of stage i, coset j

Figure 5.2. The s Stages Fast Galois Field Transform System

$p_s - 1$ in stage two, and on p_1 in the last stage s. At stage k of the transform pipeline, the intermediate variables X are grouped to b_k blocks of each size z_k, where

$$b_k = \prod_{r=1}^{s-k} p_r \tag{5.25}$$

for $s-k > 0$, and $b_k = 1$ otherwise;

$$z_k = \prod_{r=1}^{k} p_{s-k+1} \,. \tag{5.26}$$

Example 5.1: The 255-Point Good-Thomas FFT.

Let GF(p^m), with $p = 2$, $m = 8$, $d = 255 = p_1 \cdot p_2 \cdot p_3 = 3 \cdot 5 \cdot 17$. Stage one is based on 17; stage two is based on five; and stage three is based on three. Figure 5.3 shows the FFT pipeline stages.

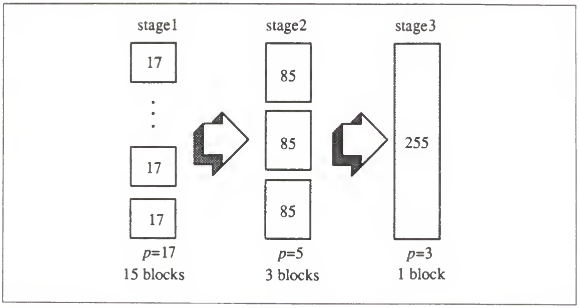


Figure 5.3 The 255-point Good-Thomas FFT Pipeline Stages

In order to evaluate the operations required for the transform, the number and the length of the cosets for each stage are counted. At the output stage, $d = 255$, the divisors of d are 255, 85, 51, 17, 15, 5, 3, and 1. Since numbers 255, 85, 51, 17 are all divisors of $2^8 - 1$, there exist 30 cosets of length eight ($\phi\ (255)/8 + \phi\ (85)/8 + \phi\ (51)/8 + \phi\ (17)/8 = 30$). Similarly, 15 and five divides $2^4 - 1$ and $\phi\ (5) = 4$ which yields three cosets of length four. The remaining three and one yield one each of length one and two. Hence, the total number of the cosets in the stage is 35. At stage two, the intermediate variables are grouped into three blocks of size 85. Each of the 85 point blocks has 10 cosets of length eight, and one each of length one and four. Hence, the total number of the cosets in the stage is 12. At stage one, there are 15 blocks of size 17, each block has two cosets of length eight and one of length one. Thus, the total number of cosets in the stage is 3. Using the properties of cyclotomic coset and the normal basis representation, the complexity analysis between conventional and normal basis approaches, in terms of the number of operations (multiplications and additions involved), is given in Table 5.1. ♦

Table 5.1 The Complexity Comparison Between Conventional and Normal Basis Approaches

method / stages	conventional	normal basis
1	$15 \cdot 17 \cdot (17-1) = 4080$	$15 \cdot 3 \cdot (17-1) = 720$
2	$3 \cdot 85 \cdot (5-1) = 1020$	$3 \cdot 12 \cdot (5-1) = 144$
3	$1 \cdot 255 \cdot (3-1) = 510$	$1 \cdot 35 \cdot (3-1) = 70$
total	5610	934

5.2.3 System Performance Analysis and Discussions

A comparison between the total number of operations involved for each method (Table 5.1) in their respective approach to the direct DFT which required approximately 65,000 operations clearly suggests a dramatic reduction in computational operations when the normal basis system is implemented. This system takes advantage of the multiplication-free shifting by applying the normal basis representation to the variables in the pipeline stages. In terms of physical realities, the simplified computational structure makes this system readily adaptable to a compact device without relinquishing any of its efficiency. Interfacing with pre-existing systems which have representations in a different format, such as the power form or the standard polynomial form, simply involves making a basis change at the end of the processing pipeline.

For the implementation of prime factor FFT over the finite field GF(p^m), p and m have to be properly chosen to meet the number of input data samples and their range. There are some general guidelines to assist the development of an efficient system.

1) The dynamic range of the data sequences to be transformed is p, which is the characteristic of the finite field GF(p^m), according to Blahut's theory. For example, for 8-bit system, p should be a prime ≤ 256; 16-bit system p a prime $\leq 65{,}536$;

2) The length of the data sequences d is a factor of $p^m - 1$;

3) To be suitable for applying FFT algorithms, the length d should be a highly composite;

4) To effectively utilize the conjugacy property of field elements, we should properly choice d and its factors such that less cosets with larger length are obtained;

5) The possible lengths of cosets are the factors of m. For example, for GF(2^8) the possible lengths of the cosets are 8, 4, 2, 1. Similarly, the possible length in GF(2^5) are 5 and 1. Thus, it is better to have larger m (or even prime m);

6) Redinbo [Rad86] suggested that the order of stages in the prime factor FFT is so essential that it will determine the complexity of the system. For example, for GF(2^8), let $d = 255$, the factors of d are 3, 5, 17. For the case of $p_1 = 3$, $p_2 = 5$, $p_3 = 17$, the operations needed for each stage are:

Stage 1 (p_3) :	$15 \cdot 3 \cdot (17 - 1) = 720$
Stage 2 (p_2) :	$3 \cdot 12 \cdot (5 - 1) = 144$
Stage 3 (p_1) :	$1 \cdot 35 \cdot (3 - 1) = 70$
Total operations :	934.

However, for $p_1 = 17$, $p_2 = 5$, $p_3 = 3$, the operations needed for each stage are:

Stage 1 (p_3) :	$85 \cdot 2 \cdot (3 - 1) = 340$
Stage 2 (p_2) :	$17 \cdot 5 \cdot (5 - 1) = 340$

Stage 3 (p_1) : $1 \cdot 35 \cdot (17 - 1) = 560.$

Total operations : 1240;

7) It would be nice to have all intermediate results in the transform be always in normal basis representation. However, the finite field multiplication of r^i and field elements during the evaluation of intermediate variables results in non-normal basis representations. This leads some extra work to convert back to normal basis. To solve this problem, r^i has to be represented in normal basis and apply normal-basis multipliers or basis conversions at the end of the evaluation should be applied. The advantage of normal basis multiplier is its simplicity and regularity; but it might take lots of space to implement the multipliers for the evaluation of intermediate variables. On the other hand, digit shift and digit-wise addition are simply performed then conversion from primal basis to normal basis is applied. This conversion can be drastically simplified if a self-dual normal basis to represent field elements can be found and used [Lid86].

5.3 The Basis-change Algorithm for Fast Finite Field Transforms

As fast Fourier transform (FFT) algorithms are applied over the finite field GF(p^m), the intermediate results are encoded in a normal basis representation. A significant computational reduction is obtained due to the conjugacy relationship within a cyclotomic coset of the intermediate variables, and the cyclic shifting property of p powers of the variables in a normal basis representation. However, the computation of each element within the conjugate cosets is still an intensive multiplicative task. To mitigate the problem, a fast and compact computational structure based on the basis-change algorithm is used to perform the multiply-accumulate operations over a finite field. Once the butterfly operation module of a cyclotomic coset based on the algorithm is developed, the integration of the modules leads to an efficient VLSI implementation of the pipeline stages of the fast transform.

While this transform can be expedited by traditional FFT-type algorithms [McC79], its computational complexity remains O(d log$d$). The concept of applying the conjugacy property and the normal basis representation of finite field elements to the transform to reduce its complexity was introduced by Kao, Taylor [Kao90] and Redinbo [Red86]. However, under normal basis representations, the evaluation of a field element of a cyclotomic coset within finite field transforms introduces a nontrivial multiply-accumulate task which may results in an extensive application of the conventional finite field multipliers (see Chapter 3).

This research effort suggests a new computational algorithm which involves the change of basis of field elements during the nontrivial evaluation of the coset elements. This finite field arithmetic is equipped with cyclic shifting for scaling, simple table lookups for polynomial reduction, and a series of component-wise modular adders. Application of these concepts to the existing fast transform not only serves to simplify the transform, but also leads to the development of a very compact signal processing device which can be implemented as a building block and seamlessly integrated to a discrete Fourier transform (DFT) system.

5.3.1 The Conjugate Sets in a Fast Finite Field Transform

The general form of a factor-type fast finite field transform V_j which is based either on the Cooley-Turkey or the Good-Thomas algorithm at the stage k can be expressed as

$$V_j = \sum_{i_{k+1}=0}^{d_{k+1}-1} (r^{c_{k+1}}) \; [\; \sum_{i_k'=0}^{p_k-1} (r^{c_k}) \; X_{j_1, j_2', \ldots, j_{k-1}', i_k} \;]$$

$$= \sum_{i_{k+1}=0}^{d_{k+1}-1} (r^{c_{k+1}}) \; X_{j_1, j_2', \ldots, j_k', i_{k+1}} \qquad (5.27)$$

where c_k and c_{k+1} are functions of i and j, p_k is a factor of d, and X is an intermediate result of the transform pipeline.

Since the intermediate results of the fast transform are the elements of the finite field GF(p^m), the conjugacy property is applicable and the transform process is simplified. The conjugacy property is demonstrated by raising the pth power of the defining items $X_{j_1, j'_2, \dots, j'_k, i_{k+1}}$ in Equation (5.27) for the intermediate results from stage k

$$
\begin{aligned}
(X_{j_1, j'_2, \dots, j'_k, i_{k+1}})^{p^t} &= \sum_{i'_k=0}^{p_k-1} (r^{d/p_k})^{i'_k j'_k p^t (d_{k+1} D_{k+1})} X^{p^t}_{j_1, j'_2, \dots, j'_{k-1}, i_k} \\
&= \sum_{i'_k=0}^{p_k-1} (r^{d/p_k})^{i'_k R'_{k,t} (d_{k+1} D_{k+1})} X_{R_{1,t}, R'_{2,t}, \dots, R'_{k-1,t}, i_k} \\
&= X_{R_{1,t}, R'_{2,t}, \dots, R'_{k,t}, i_{k+1}} \qquad (5.28)
\end{aligned}
$$

where $t = 1, 2, \dots, l$, and l is a coset length.

A modularized circuitry for the evaluation of a conjugate set is developed accordingly and demonstrated in Figure 5.4. Generation of the l intermediate variables occurs in the evaluation module where there is only one variable which needs to be evaluated. The operations include $p_k - 1$ modular multiplications and additions which are performed in the multiply-accumulate unit (MAU). The remaining $l-1$ variables are obtained by the simultaneous cyclic shiftings of the evaluated variable. The design consists of $p_k - 1$ normal basis multipliers, a p_k-input m-digit component-wise modular adder, a cyclic shift routing network, and l, m-digit registers for result storage.

The fast transform algorithm dramatically reduces computational complexity. However, the evaluation of the leading element in a coset is still a rigorous multiplicative task which involves normal basis multiplications of a previous stage variable X_i and a primitive element r^{ig} as is shown in Equation (5.27).

5.3.2 The Basis-Change Algorithm

To maintain the inherent advantage of the cyclic shifting within a normal basis scheme, the intermediate variable evaluated in Equation (5.27) must remain in the normal basis form. However, the result of the multiply-accumulate operations always deviates from this rule.

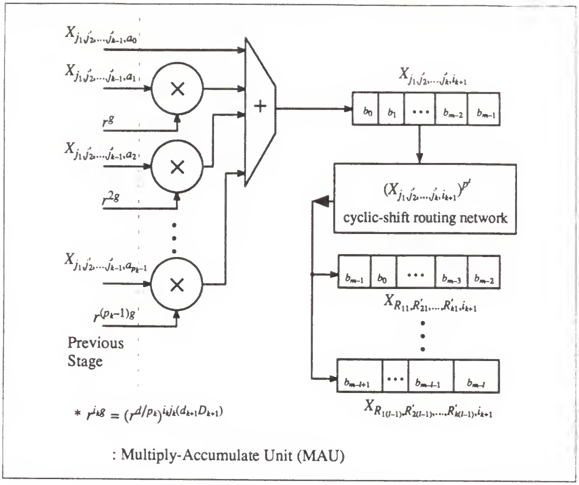

$X_{j_1,j'_2,\ldots,j'_{k-1},a_0}$
$X_{j_1,j'_2,\ldots,j'_{k-1},a_1}$
r^g
$X_{j_1,j'_2,\ldots,j'_{k-1},a_2}$
r^{2g}
$X_{j_1,j'_2,\ldots,j'_{k-1},a_{p_k-1}}$
$r^{(p_k-1)g}$
Previous
Stage
$X_{j_1,j'_2,\ldots,j'_k,i_{k+1}}$
b_0 b_1 $\cdots$ b_{m-2} b_{m-1}
$(X_{j_1,j'_2,\ldots,j'_k,i_{k+1}})^{p^l}$
cyclic-shift routing network
b_{m-1} b_0 $\cdots$ b_{m-3} b_{m-2}
$X_{R_{11},R'_{21},\ldots,R'_{k1},i_{k+1}}$
b_{m-l+1} $\cdots$ b_{m-l-1} b_{m-l}
$X_{R_{1(l-1)},R'_{2(l-1)},\ldots,R'_{k(l-1)},i_{k+1}}$
$* \; r^{i_k g} = (r^{d/p_k})^{i_k j_k (d_{k+1} D_{k+1})}$
: Multiply-Accumulate Unit (MAU)

Figure 5.4 The Conjugate Set Evaluation Circuit

The variables $X_{i,k-1}$ in normal basis representation from stage k–1 with the multiplication of r^{ig} are represented explicitly as

$$X_{i,k-1} r^{ig} = (a_{i,0} r + a_{i,1} r^{p} + a_{i,2} r^{p^2} + \ldots + a_{i,m-1} r^{p^{m-1}}) r^{ig} \tag{5.29}$$

where $i = 0, 1, \ldots, p_k - 1$. The variable $X_{i,k}$ of stage k which is the sum of Equation (5.29) for all i becomes an "unreduced" polynomial of possible degree of $n = p^{m-1} + (p_k - 1)g$ and is represented as

$$X_{i,k} = c_0 r + c_1 r^{q_1} + \ldots + c_n r^{q_n} \tag{5.30}$$

where c_s is a function of $a_{i,j}$ for $s = 0, 1, \ldots, n$ and q_t is a function of $(p^e + fg)$ for $t = 0, 1, \ldots, n$. Obviously, Equation (5.30) is not in a form of the normal basis from which the remaining variables in the same coset can be obtained by the cyclic shift property. As a result, a novel algorithm capable of performing the basis-change process without dealing with the normal basis multiplications is presented as follows:

1) Reduce the exponent of r^{q_t} where $q_t > p^m-1$ by performing modulo (p^m-1) in accordance with Fermat's theorem which states $r^{p^m-1} \equiv 1$;

2) Reduce the exponent of r^{q_t} where $q_t > m-1$ by expressing r^{q_t} in terms of polynomials modulo the field-defining polynomial for GF(p^m). This may be carried out simply by th exponent table lookup operation;

3) Rearrange the result of step 2) which may involve trivial multiplications and additions within the ground field GF(p) to form the primal basis representation of the variable $X_{i,k}$;

4) Build lookup tables of the normal basis representations for each component of the primal basis vector $[\ 1, r, r^2, \ldots, r^{m-1}\ ]$; and

5) Convert the primal basis variable in step 3) to the normal basis form with the assistance of the lookup table in step 4). These operations only involves additions within the ground field GF(p).

5.3.3 System Development

The MAU subsystem in Figure 5.4 is further developed according to the discussion in the preceding section. The architecture of the MAU device is subdivided into four processing elements in a pipeline fashion. These elements include the Scaling Shifter, the Vector Combiner, the Primal Basis Generator, and the Normal Basis Generator — all of which are described as follows:

1) Scaling Shifter: the multiplication of a intermediate variable $X_{i,k-1}$ and the factor r^{jg} as found in Equation (5.29) is equivalent to a feedback shifting implementation. This is a

direct consequence of the fact that $r^{p^m-1} \equiv 1$. Figure 5.5 provides a pictorial demonstration of the scaling shifting operation where $r^{p^{m-1}+ig} \equiv r^1$ for the case of $p^{m-1} + ig = p^m$;

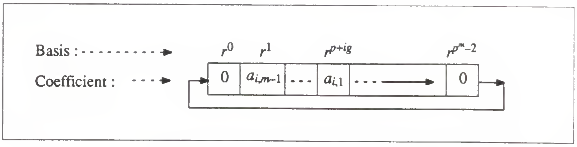


Figure 5.5 An Example of the Scaling Shifter

2) Vector Combiner: the outputs of the Scaling Shifters are all represented as vectors of the basis in the form of [$1, r, r^2, \ldots, r^{p^m-2}$]. This means that the sum of these vector may involve component-wise addition in the field GF(p). A symbolic representation of the vector combining unit is depicted in Figure 5.6;

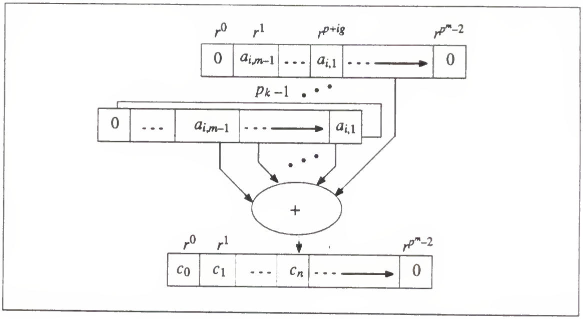


Figure 5.6 An Example of the Vector Combiner

3) Primal Basis Generator: the (p^m–1)-digit output of the vector combiner shown in Figure 4 needs to be changed to a regular primal basis of (m–1)-digit. The preprogrammed lookup tables — primal lookup table (PLUT) — is built for polynomial reduction where dig-

its in the vector (polynomial) have a corresponding exponent greater than $m-1$. The table is computed using a discrete exponentiation method where the exponent i is presented to the address of the table and points accordingly to the content of the memory. The relationship of an exponent i and its corresponding primal basis form is expressed as

$$\exp_r(i) = r^i = d_0' + d_1' r + d_2' r^2 + \ldots + d_{m-1}' r^{m-1} .$$

A primal basis representation of the variable $X_{i,k}$ is obtained by table-lookups and additions over GF(p). A functional block is shown in Figure 5.7; and

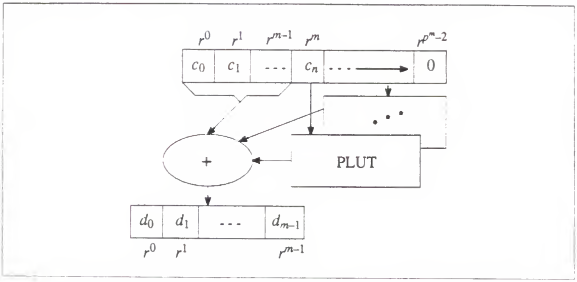


Figure 5.7 The Primal Basis Generator

4) Normal Basis Generator: normal basis representations of the powers of r are precomputed, then stored in the lookup table which is called the normal lookup table (NLUT). The address, j, and the content of the table memory have the relation

$$r^j = d_0' r + d_1' r^p + d_2' r^{p^2} + \ldots + d_{m-1}' r^{p^{m-1}} .$$

A similar architecture to the primal basis generator is developed for the normal basis conversion and is demonstrated in Figure 5.8.

5.3.4 System Complexity Evaluation and Example

The system complexity, in terms of hardware implementation cost, for the four processing elements discussed in the previous section is summarized as follows:

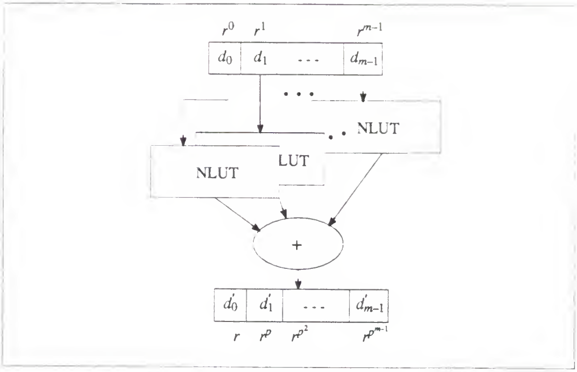


Figure 5.8 The Normal Basis Generator

1) Scaling Shifter: although it is a multiplication of an element of the normal basis and a factor r^{ig}, the primary operation is simply a matter of exponential arithmetic. Hence, m modulo (p^m-1) adders are required for each one of the (p_k-1) channels where the processing latency is the delay of an adder;

2) Vector Combiner: this operation is equivalent to the addition of (p_k-1) vectors of basis length of p^m-1. The worse case data path requires (p_k-2) modulo-p adders with the latency translated to $\lceil \log_2(p_k-1) \rceil$ levels of the adder operation (Note: $\lceil . \rceil$ represents the smallest integer $\geq t \in \mathbf{R}$.);

3) Primal Basis Generator: the conversion of the digit within the vector to the primal basis requires at most $(p_k-1)m$ memory tables with the configuration of p by $m(\log_2 p)$. The worse case data path requires m channels of addition which, in turn, requires $(p_k-1)m-1$ modulo-p adders with the latency of $\lceil \log_2 [\ (p_k - 1\ )\ m\ ] \rceil$ levels of the adder operation; and

4) Normal Basis Generator: similarly, the conversion of the digit in a primal basis vector to the normal basis requires m memory tables with the configuration of p by $m\lceil \log_2 p \rceil$. The worse case data path requires m channels of addition which requires $(m-1)$ modulo-p adders with the latency of $\lceil \log_2 (m-1) \rceil$ levels of the adder operation.

Example 5.2: The Basis-Change algorithm.

Let the finite field GF(p^m) where $p = 2$ and $m = 4$ have a defining polynomial $f(x) = x^4 + x^3 + 1$. The set of roots { r, r^2, r^4, r^8 } constitutes a normal basis of GF(2^4). Assume in one stage of a factor-type transform $p_k = 3$ and $r^{ig} = r^{(i5)}$ for $i = 0, 1, 2$; and the set of the previous stage variables in the normal basis representation are { $X_{0,k}$, $X_{1,k}$, $X_{2,k}$ } = { r, r^3, r^7 } (power form) = { (1000), (1101), (0011)} (normal basis). Referring to Equation (5.27), the evaluation of the leading element $X_{0,k+1}$ in a conjugate coset is to evaluate the following sum-of-product in the MAU as

$$X_{0,k+1} = \sum_{i=0}^{2} (r^{5i}) X_{i,k} = X'_{0,k} + X'_{1,k} + X'_{2,k} .$$

Applying the basis-change algorithm:

Step 1. (Scaling shifting):

$$X_{0,k}\, r^0 = r;$$

$$X_{1,k}\, r^5 = r^3 r^5 = r^8 = r^6 + r^7 + r^{13};$$

$$X_{2,k}\, r^{10} = r^7 r^{10} = r^{17} = r^{14} + r^{18} \rightarrow r^{14} + r^{3 \equiv \text{mod } 15} ;$$

Step 2. (Vector combining):

$$X'_{0,k} + X'_{1,k} + X'_{2,k} = r + r^3 + r^6 + r^7 + r^{13} + r^{14};$$

Step 3. (Primal basis generation):

$$X_{0,k+1} = r + r^3 + (r^3 + r^2 + r + 1) + (r^2 + r + 1) + (r^2 + r) +$$

$$(r^3 + r^2) = r^3 ; \text{ and}$$

Step 4. (Normal basis generation):

$X_{0,k+1} = r^3$ (primal basis) $\rightarrow$ (1101) (normal basis).

The operations needed in the algorithm are five modulo 15 additions in Step 1; no operation in Step 2; four PLUT lookups (for r^6, r^7, r^{13}, and r^{14}) and six modulo 2 additions in Step 3; and one NLUT lookup in Step 4. ♦

This algorithm enables a simple and fast evaluation of a leading element by applying exponent arithmetic to the field element scaling, simple table lookups, and minor modular additions over the field GF(p). The conventional approach might involve the linear feedback shift register (LFSR) type finite field multipliers [Lid86 Chapter 6] with slow-clocking latency or the bulky parallel normal basis multiplier [Wan85]. This module performs as a fundamental building block, along with a routing network which accomplishes barrel cyclic shiftings for the remaining elements in a conjugate coset, and is repeatedly integrated to each intermediate stage of a factor-type FFT system. Consequently, this module is an economical, high-performance VLSI component.

CHAPTER 6
THE PIPELINE POLYNOMIAL RNS PROCESSOR WITH FERMAT NUMBER TRANSFORM

6.1 Introduction

The application of Polynomial Residue Number Systems (PRNS's) in complex multiplication offers tremendously low complexity within the digital signal processing (DSP) area. Unfortunately, isomorphic mappings between complex number and PRNS domains suffer from a nontrivial transform problem which eventually precludes the inherent advantages of the PRNS approach. A significant simplification in the mapping procedure is achieved by a FFT-like scheme and a sequence of primitive split-then-add operations. These operations originate from an algebraic congruence and a residue reduction of a Fermat prime within finite fields. An efficient custom VLSI implementation of the FFT-type multiplier-free system confirms the advantages of the novel mapping algorithm.

The complex arithmetic of residue number systems in DSP applications is a recent subject of intense study. Cozzens and Finkelstein [Coz85] suggest an improved algorithm for complex number approximations using algebraic integers in higher degree extensions of the rational number **Q**. The PRNS provides a special mechanism for complex number operations[Ska87]. Although it suggests both performance and implementation advantages, the isomorphic mapping structure between the complex domain and PRNS domain results in an awkward transform problem. This research effort demonstrates a solution to the mapping procedure difficulties by applying the algebraic congruence concept and the residue reduction technique.

Polynomial factorization over finite fields along with the polynomial version of the Chinese remainder theorem (CRT) are the essential elements of PRNS system development.

The forward and inverse isomorphic mappings are shown to be parallel to the discrete Fourier transform (DFT). Given an Nth degree polynomial $f(x) = x^N + 1$, the equation $f(x) = 0$ is solved by applying the algebraic congruence of Fermat's theorem. Then, the primitive roots r_i where $r_i^N \equiv -1 \bmod M$ and $\phi\,(M) = 2N$, ϕ is Euler's totient function, are found. Consequently, a FFT-type fast mapping is obtained based upon the relationship of the primitive roots. Since the basic operations in fast mapping are the scalings, $x\,r_i^k$, a novel technique is applied to residue reduction which, in turn, simply translates the scaling operations into trivial shift-then-adds. As a result, the two-level reduction in complexity becomes a reinforcement of the PRNS system.

A prototype VLSI design of a five-bit multiplier-free FFT-type PRNS processor has been implemented using the Magic IC layout tool. This design consists of 13K transistors with a 6.1 by 6.0 square millimeter footprint. The device also has complete logic and timing simulated on the HP-DCS design tool. In terms of speed, cost and simplicity, the innovative approach of this new design outperforms the conventional systems in current usage.

6.1.1 Reasons To Use Fermat Prime Number As M

Let N is the least positive integer, such that $\alpha^N \equiv 1 \bmod M$, and M defined by one of the following cases.

1) If M is even, that is M having a factor of 2, therefore, the maximum value of N is 1. This is not a promising result that implies M should be odd.

2) If $M = 2^K - 1$, K a composite number pq, where p is a prime, then

$$2^{pq} - 1 = (2^p - 1)(2^{p(q-1)} + 2^{p(q-2)} + \ldots + 2^p + 1), \qquad (6.1)$$

such that $2^p - 1 \mid M$. This means the maximum possible length of the transform N will be governed by the length possible for $2^p - 1$.

3) If M is a *Mersenne* number of the form $2^K - 1$, where K is prime, Rader [Rad86a] showed that transforms of length at least $2K$ exist and the corresponding α is -2. Because $2K$ is not

highly composite and therefore, there does not exist a fast FFT-type implementation algorithm.

4) If $M = 2^K + 1$ and K odd, then 3 divides M and the largest positive transform length N is 2.

5) If $M = 2^K + 1$ and K even, for $K = m2^n$, where m is odd, then $2^{2^n} + 1 \mid M$ and the length of the possible transform will be governed by the transform length possible for $2^{2^n} + 1$. These number are known as Fermat numbers which are opportunities for FFT-type algorithms.

6.1.2 Algebraic Congruence to a Fermat Prime

Let M be a Fermat prime where M is of the form of $2^m + 1, m = 2^n, n \le 4$. According to Fermat's theorem, $x^{M-1} \equiv 1 \bmod M$ for all nonzero $x \in \mathbf{Z}_M$, that is

$$x^{2^m} \equiv 1 \bmod M. \tag{6.2}$$

The nonzero elements in the group $\mathbf{Z}_M$ are roots of Equation (6.2). Thus, the group can be generated by a primitive root α. Since the order of the group is $\phi(M) = 2^m$, the set of the roots of Equation (6.2) is constructed as

$$R = \{ \alpha^1, \alpha^2, \ldots, \alpha^{2^m} = \alpha^0 \}. \tag{6.3}$$

This means Equation (6.2) can be factored over the ring as

$$x^{2^m} - 1 = \prod_{i=0}^{2^m-1} (x - \alpha^i). \tag{6.4}$$

Equation (6.2) also indicates

$$(x^{2^m})^{\frac{1}{2}} \equiv \pm 1 \bmod M. \tag{6.5}$$

The roots of Equation (6.4) can be analyzed for different cases. For the case where $x^{2^{m-1}} \equiv 1 \bmod M$, each x can not be the primitive roots; otherwise, the definition of primitive root is contradicted. This leads to the fact that x is all elements except the primitive roots; that is $x \in R = \{ \alpha^i \mid (i, 2^m) = 2^1, 2^2, \ldots, 2^{m-1} \}$. On the other hand, for $x^{2^{m-1}} \equiv -1 \bmod M \equiv 2^m \bmod M$,

x is a member of the primitive roots which are $R = \{ \alpha^i \mid (i, 2^m) = 1 \}$. This means Equation (6.5) can be factored, over the group, as

$$x^{2^{m-1}} + 1 = \prod_{i=0}^{2^{m-1}-1} (x - \alpha^{2i+1}) . \tag{6.6}$$

Equation (6.6) states that in the case of $N = 2^{m-1}$ the roots are primitive roots (of order 2^m) modulo M and are of the form of α^{2i+1} for $i = 0, 1, \ldots, N-1$. For the case of $N = 2^{m-2}$ the roots of equation $x^N + 1 \equiv 0 \bmod M$ are obtained by first finding a root α of order 2^{m-1}, and subsequently raising it to the power $2i + 1$ for $i = 0, 1, \ldots, N-1$. A similar procedure applies to the sequel where $N = 2^{m-k}$ for $2 < k < m$, k an integer.

According to the above discussion, the roots, r_i, of the equation $x^N \equiv -1 \bmod M$, where $N \mid 2^{m-1}$, are solved and represented as follows:

$$r_0 = 2^{\frac{m}{N}} \bmod M ,$$

$$r_1 = 2^{\frac{3m}{N}} \bmod M = r_0^3$$

$$\vdots$$

$$r_i = 2^{\frac{(2i+1)m}{N}} \bmod M = r_0^{2i+1} , \tag{6.7}$$

where $i = 0, 1, \ldots, N-1$. Furthermore, an additional relationship between roots is obtained according to the following example:

$$\begin{aligned} r_{N-1} &= 2^{\frac{(2(N-1)+1)m}{N}} \bmod M \\ &= (2^m)^2 \; 2^{-(\frac{m}{N})} \bmod M \\ &= (-1)^2 \; 2^{-(\frac{m}{N})} \bmod M \\ &= r_0^{-1}. \end{aligned} \tag{6.8}$$

Further result indicates that $r_{N-2} = r_0^{-3}$. Consequently, a general form is obtained

$$r_{N-i} = r_0^{-(2i-1)} , \tag{6.9}$$

where $i = 1, 2, \ldots, N$.

Example 6.1 The roots of $x^N \equiv -1 \bmod M$.

Let $N = 4$ and M be a Fermat prime, then $r_1 = r_0^3$, $r_2 = r_0^5$, and $r_3 = r_0^7$. From Equation (6.9), these roots are also rewritten as $r_3 = r_0^{-1}$, $r_2 = r_0^{-3}$, and $r_1 = r_0^{-5}$. ♦

6.1.3 The Multiplier-Free Fast FFT-Type Isomorphic Mappings

The computational complexity and hardware cost for the multiplication of two polynomials is reduced to the cost of introducing the isomorphic mappings found in Chapter 4 Equations (4.35) and (4.36). The simplification of these nontrivial mappings is the major task in this section. Two methods are introduced — namely, FFT-type transform and multiplier-free scaling.

The Fast FFT-Type Algorithm. The FFT-type algorithms are applied to the mappings based on the facts that $M - 1$ is highly composite, and the property which $r^N \equiv -1 \bmod M$ where $r \in \mathbf{Z}_M$. Furthermore, the relationship between the roots of Equation (6.6) is also referred.

Let the input and output variables of the isomorphic mapping are a_n and A_k, respectively, where $n, k = 0, 1, \ldots, N-1$. Analogous to the Cooley-Turkey FFT algorithm, the isomorphic mappings are derived as follows:

1) Isomorphic Forward Transform

From Equation (4.35), the PRNS format of the input variable a_n is

$$A_k = \sum_{n=0}^{N-1} a_n r_k^n . \tag{6.10}$$

Since Equation (6.7) implies that $r_k = r^{2k+1}$, Equation (6.10) becomes

$$A_k = \sum_{n=0}^{N-1} a_n r^{n(2k+1)} . \tag{6.11}$$

To form the FFT-type Transform, let a_n be separated into its even- and odd-numbered points as follows:

$$A_k = \sum_{n\ even} a_n r^{n(2k+1)} + \sum_{n\ odd} a_n r^{n(2k+1)} \quad . \tag{6.12}$$

If a substitution of variables $n = 2d$ for n even and $n = 2d + 1$ for n odd is made, then Equation (6.12) becomes

$$A_k = \sum_{d=0}^{\frac{N}{2}-1} a_{2d} r^{2d(2k+1)} + \sum_{d=0}^{\frac{N}{2}-1} a_{2d+1} r^{(2d+1)(2k+1)}$$

$$= \sum_{d=0}^{\frac{N}{2}-1} a_{2d}(r^2)^{d(2k+1)} + r^{2k+1} \sum_{d=0}^{\frac{N}{2}-1} a_{2d+1}(r^2)^{(d(2k+1)} . \tag{6.13}$$

Since $(r^2)^{\frac{N}{2}} \equiv -1 \bmod M$ and let $s = r^2$; Thus, Equation (6.13) can be rewritten as

$$A_k = \sum_{d=0}^{\frac{N}{2}-1} a_{2d} s^{d(2k+1)} + r^{2k+1} \sum_{d=0}^{\frac{N}{2}-1} a_{2d+1} s^{(d(2k+1)}$$

$$= A_k' + r^{2k+1} A_k'' . \tag{6.14}$$

Each of the sums in Equation (6.14) is recognized as an $N/2$-point transform. After the two transforms are computed, they are combined to yield the N-point transform A_k. This procedure continues until N is completely decomposed; hence, $\log_2 N$ times iterations are required to complete the transforms.

2) The Isomorphic Inverse Transform

From Equation (4.36), the regular format of the variable A_k is

$$a_n = N^{-1} \sum_{k=0}^{N-1} A_k r_k^{-n} . \tag{6.15}$$

Since Equation (6.7) implies that $r_k = r^{2k+1}$, Equation (6.10) becomes

$$a_n = N^{-1} \sum_{k=0}^{N-1} A_k r^{-n(2k+1)} \qquad (6.16)$$

where N^{-1} is the multiplicative inverse of N and $n = 0, 1, \ldots, N-1$. Then A_k is separated into its even- and odd-numbered points:

$$a_n = N^{-1}\Big(\sum_{k\ even} A_k r^{-n(2k+1)} + \sum_{k\ odd} A_k r^{-n(2k+1)} \Big) . \qquad (6.17)$$

Similarly, if a substitution of variables $k = 2e$ for k even and $k = 2e + 1$ for k odd is made, Equation (6.17) becomes

$$a_n = N^{-1}\Big(\sum_{e=0}^{\frac{N}{2}-1} A_{2e} r^{-n(4e+1)} + \sum_{e=0}^{\frac{N}{2}-1} A_{2e+1} r^{-n(4e+3)} \Big)$$

$$= N^{-1}\Big(r^n \sum_{e=0}^{\frac{N}{2}-1} A_{2e} (r^2)^{-n(2e+1)} + r^{-n} \sum_{e=0}^{\frac{N}{2}-1} A_{2e+1} (r^2)^{-n(2e+1)} \Big) . \qquad (6.18)$$

Since $(r^2)^{\frac{N}{2}} \equiv -1 \bmod M$ and let $s = r^2$; Equation (6.18) can be rewritten as

$$a_n = N^{-1}\Big(r^n \sum_{e=0}^{\frac{N}{2}-1} A_{2e} s^{-n(2e+1)} + r^{-n} \sum_{e=0}^{\frac{N}{2}-1} A_{2e+1} s^{-n(2e+1)} \Big)$$

$$= N^{-1}(r^n a_n' + r^{-n} a_n'') . \qquad (6.19)$$

Similar formats are found in Equations (6.14) and (6.19), except for an extra scaling operation (due to N^{-1}) is required at the end of the inverse transform. Instead of performing the extra level of multiplication, a simple scaling technique merges N^{-1} to the final stage of the transform. As a result, this extra level of latency is eventually eliminated. A detailed description of the technique is shown in the following section.

The Multiplier-Free Scalings Technique. Previous discussion shows that the fundamental operations embedded in the FFT-type isomorphic mappings are the scaling operations in the form of $r_i^k a_j$ and $r_i^k A_j$ modulo M, where a_j or A_j is an isomorphic mapping pair and r_i is a root of $r^N \equiv -1 \bmod M$. By virtue of the fact that all roots r_i are in the form of binary powers (see Equation (6.7)), these scaling operations are easily facilitated to shift-then-adds instead of performed as regular modular multiplications.

A number, $x \in \mathbf{Z}_M$, is partitioned and denoted as an $n + 1$ bit word as

$$x = x_H 2^e + x_L \mapsto [\, x_H : x_L \; ; e \,] \tag{6.20}$$

where $x_H \in \mathbf{Z}_t$, $t = [\,(M-1)/2^e\,] + 1$, and $x_L \in \mathbf{Z}_{2^e}$. Hence, $a \in \mathbf{Z}_M$ is partitioned and is of the form $a = [\, a_H : a_L \; ; (m - ((2i+1)m)/N) \,]$. Consequently, $r_i\, a$ is evaluated efficiently as

$$\begin{aligned} r_i\, a \mod M &\equiv 2^{\frac{(2i+1)m}{N}} \, (a_H \, 2^{(m - \frac{(2i+1)m}{N})} + a_L) \mod M \\ &\equiv (2^m a_H + 2^{\frac{(2i+1)m}{N}} a_L) \mod M \\ &\equiv ((2^m + 1) a_H - a_H + 2^{\frac{(2i+1)m}{N}} a_L) \mod M \\ &\equiv (2^{\frac{(2i+1)m}{N}} a_L - a_H) \mod M. \end{aligned} \tag{6.21}$$

The result indicates a significant simplification of the product computation $r_i\, a$. This is achieved by partitioning, then adding (shift-then-add).

The scaling factor N^{-1} of Equation (6.19) can be treated in a similar manner. First of all, N^{-1} is evaluated by applying the definition of inverse of N, which is $NN^{-1} \equiv 1 \mod M$. Since $M = 2^m + 1$ and N is a factor of $M - 1$, then $N = 2^{m-i}$ where $i = 0, 1, \ldots, m-1$. Accordingly,

$$2^{m-i} N^{-1} \equiv 1 \mod M. \tag{6.22}$$

Furthermore, since $1 \equiv -2^m \mod M$, Equation (6.22) indicates that $N^{-1} = -2^i$. Thus, if a is partitioned to the form $a = [\, a_H : a_L \; ; (m-i) \,]$, then

$$N^{-1} a \mod M \equiv (-2^m a_H - 2^i a_L \,) \mod M \equiv a_H - 2^i a_L \mod M. \tag{6.23}$$

6.2 A Pipeline Third-order Polynomial RNS Processor

Under the consideration of acceptable data accuracy and low complexity of the system design, let $N = 4$. Hence, multiplication of two complex numbers is performed as a multiplication of two third-order polynomials modulo $x^4 + 1$. The congruence $x^4 + 1 \equiv 0 \mod M$ has four distinct solutions (r, r^3, r^5, r^7) in $\mathbf{Z}_M$. By applying FFT-type transforms and the congruence property of the roots due to the fact that $r^4 \equiv -1 \bmod M$, the second stage of the isomorphic forward transform A of a is

$$A_0 = A_0' + rA_0'';$$

$$A_1 = A_1' + r^3 A_1'';$$

$$A_2 = A_2' + r^5 A_2'' = A_2' + (-r)A_2''$$

and

$$A_3 = A_3' + r^7 A_3'' = A_3' + (-r^3)A_3''. \tag{6.24}$$

Similar procedure along with the fact that $r^6 \equiv -r^2 \bmod M$ is taken to the evaluation of the first stage. It turns out that the terms A_i', A_i'' in the first stage, where $i = 0, 1, 2, 3$, can be redefined as follows:

$$A_0' = a_0 + r^2 a_2,$$

$$A_0'' = a_1 + r^2 a_3; \tag{6.25}$$

$$A_1' = a_0 + r^6 a_2 = a_0 - r^2 a_2,$$

$$A_1'' = a_1 + r^6 a_3 = a_1 - r^2 a_3; \tag{6.26}$$

$$A_2' = a_0 + r^2 a_2 = A_0',$$

$$A_2'' = a_1 + r^2 a_3 = A_0'';$$

and

$$A_3' = a_0 + r^6 a_2 = a_0 - r^2 a_2 = A_1',$$

$$A_3'' = a_1 + r^6 a_3 = a_1 - r^2 a_3 = A_1''.$$

A two stage pipeline forward transform is demonstrated in the following block diagram in Figure 6.1.

With similar procedure along with the fact that $r^{-4} \equiv (r^4)^{-1} \equiv -1 \bmod M$ and the result in Example 6.1, the inverse transform is expressed as

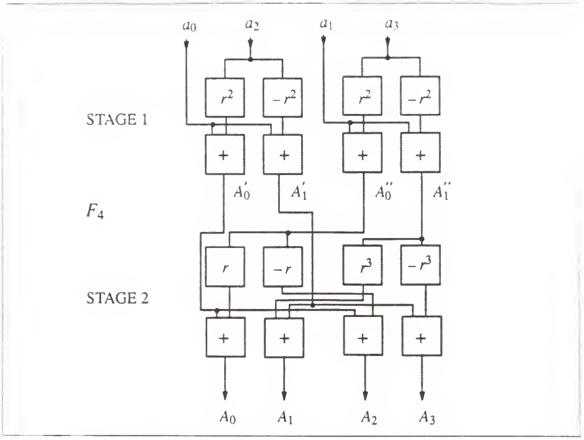


Figure 6.1 The Isomorphic Forward Transform F_4

$$a_0 = N^{-1}(A_0 + A_2 + A_1 + A_3\ );$$

$$\begin{aligned} a_1 &= N^{-1}(r^{-1}A_0 + r^{-5}A_2 + r^{-2}(r^{-1}A_1 + r^{-5}A_3)) \\ &= N^{-1}(r^{-1}(A_0 - A_2) + r^{-3}(A_1 - A_3)) \\ &= N^{-1}(-r^3(A_0 - A_2) + (-r)(A_1 - A_3)); \end{aligned}$$

$$\begin{aligned} a_2 &= N^{-1}(r^{-2}A_0 + r^{-10}A_2 + r^{-4}(r^{-2}A_1 + r^{-10}A_3)) \\ &= N^{-1}(-r^2(A_0 + A_2) + r^2(\ A_1 + A_3)); \end{aligned}$$

and

$$\begin{aligned} a_3 &= N^{-1}(A_0 r^{-3} + A_2 r^{-15} + r^{-6}(A_1 r^{-3} + A_3 r^{-15})) \\ &= N^{-1}(-r(\ A_0 - A_2) + (-r^3)(A_1 - A_3)). \end{aligned} \tag{6.27}$$

Since the extra scaling, N^{-1}, a three stage pipeline inverse transform is demonstrated in the following block diagram in Figure 6.2.

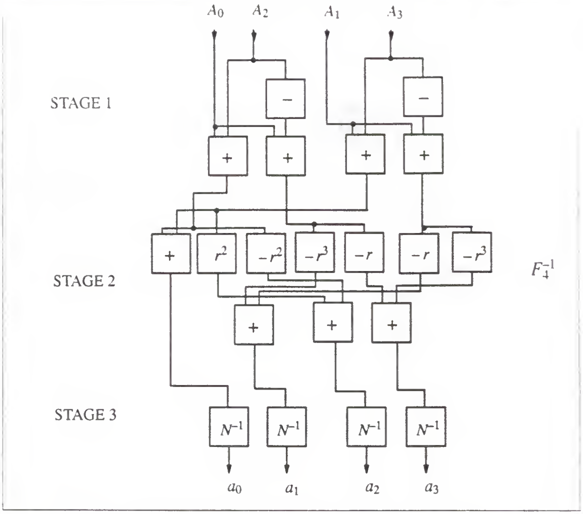


Figure 6.2 The Isomorphic Inverse Transform F_4^{-1}

Using the multiplier-free scaling techniques discussed in previous section, the scaling factors such as $\pm r$, $\pm r^2$, and $\pm r^3$ are derived as follows:

1) Scaling by r and $-r$: Let

$$a = [\ a_H : a_L ; (3/4)m\ ]$$

such that

$$ra \bmod M = 2^{m/4}\ a_L - a_H \tag{6.28}$$

and

$$-ra \bmod M = -2^{m/4}\ a_L + a_H . \tag{6.29}$$

2) Scaling by r^2 and $-r^2$: Let

$$a = [\ a_H : a_L ; (1/2)m\ ]$$

such that

$$r^2 a \bmod M = 2^{m/2}\ a_L - a_H \tag{6.30}$$

and

$$-r^2 a \bmod M = -2^{m/2}\ a_L + a_H. \tag{6.31}$$

3) Scaling by r^3 and $-r^3$: Let

$$a = [\ a_H : a_L ; (1/4)m\ ]$$

such that

$$r^3 a \bmod M = 2^{3m/4}\, a_L - a_H \tag{6.32}$$

and

$$-r^3 a \bmod M = -2^{3m/4}\, a_L + a_H. \tag{6.33}$$

A pictorial representation of these scaling module is shown in Figure 6.3.

A nine-bit PRNS arithmetic prototype is implemented based upon the theoretical development where $N = 4$, $M = 2^8 + 1$, hence, $N^{-1} = -2^{m-2} = -2^6 = -r^3$. As a result, the three-stage pipeline in the inverse transform can be reduced to two stages by the following procedure. Referring to Equation (6.27) where N^{-1} is replaced by $-r^3$, the transformed variables a_i become

$$\begin{aligned}
a_0 &= -r^3 (A_0 + A_2 + A_1 + A_3\ ); \\
a_1 &= -r^3 (r^{-1}A_0 + r^{-5}A_2 + r^{-2}(r^{-1}A_1 + r^{-5}A_3)) \\
&= -r^2 (A_0 + r^{-4}A_2) + (-A_1 - r^{-4}A_3) \\
&= -r^2 (A_0 - A_2) + (A_3 - A_1); \\
a_2 &= -r^3 (r^{-2}A_0 + r^{-10}A_2 + r^{-4}(r^{-2}A_1 + r^{-10}A_3) \\
&= -r(A_0 + A_2) + r(\ A_1 + A_3);
\end{aligned}$$

and

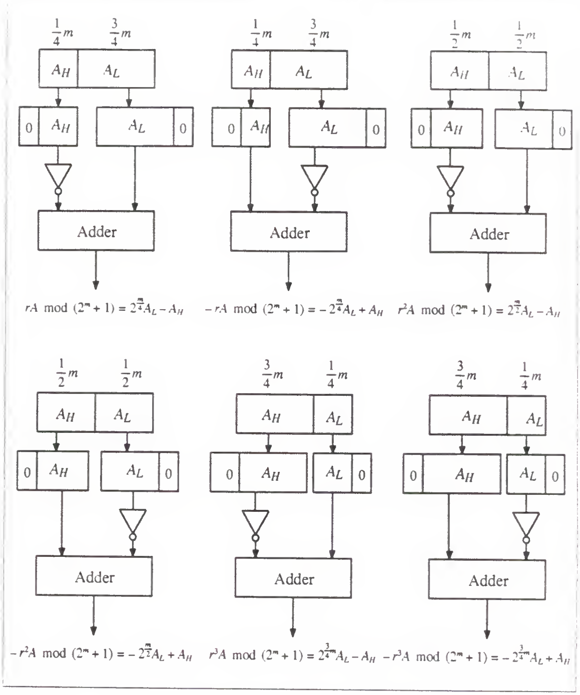

$\frac{1}{4}m$ $\frac{3}{4}m$
A_H A_L
0 A_H A_L 0
Adder
$rA \bmod (2^m+1) = 2^{\frac{m}{4}}A_L - A_H$
$\frac{1}{4}m$ $\frac{3}{4}m$
A_H A_L
0 A_H A_L 0
Adder
$-rA \bmod (2^m+1) = -2^{\frac{m}{4}}A_L + A_H$
$\frac{1}{2}m$ $\frac{1}{2}m$
A_H A_L
0 A_H A_L 0
Adder
$r^2A \bmod (2^m+1) = 2^{\frac{m}{2}}A_L - A_H$
$\frac{1}{2}m$ $\frac{1}{2}m$
A_H A_L
0 A_H A_L 0
Adder
$-r^2A \bmod (2^m+1) = -2^{\frac{m}{2}}A_L + A_H$
$\frac{3}{4}m$ $\frac{1}{4}m$
A_H A_L
0 A_H A_L 0
Adder
$r^3A \bmod (2^m+1) = 2^{\frac{3}{4}m}A_L - A_H$
$\frac{3}{4}m$ $\frac{1}{4}m$
A_H A_L
0 A_H A_L 0
Adder
$-r^3A \bmod (2^m+1) = -2^{\frac{3}{4}m}A_L + A_H$

Figure 6.3 The Isomorphic Mapping Scalar Modules

$$a_3 = -r^3(A_0 r^{-3} + A_2 r^{-15} + r^{-6}(A_1 r^{-3} + A_3 r^{-15})$$

$$= (A_2 - A_0) + (-r^2)(A_1 - A_3). \qquad (6.34)$$

A two stage pipeline inverse transform is demonstrated in the following block diagram in Figure 6.4 .

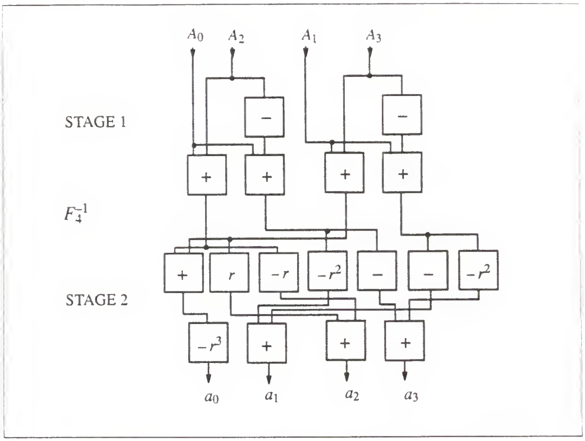


Figure 6.4 The Isomorphic Inverse Transform F_4^{-1}

A two-stage pipeline 9-bit third-order PRNS system is developed and shown in Figure 6.5 where the processing module — AMmodM_x is a modular adder and multiplication unit.

6.3 System Implementation of the Third-order Polynomial RNS Processor

According to the theoretical background developed in previous section, a third-order polynomial RNS processor is implemented using various design technologies. A two-level hierarchical implementation approach is taken to ensure the effectness and accuracy of the design. The top level of the design flow is a functional verification which includes logic simulation and unit-delay timing simulation. The lower level of the processor design is a topological layout of transistors which is a physical implementation of the design. A trade-off

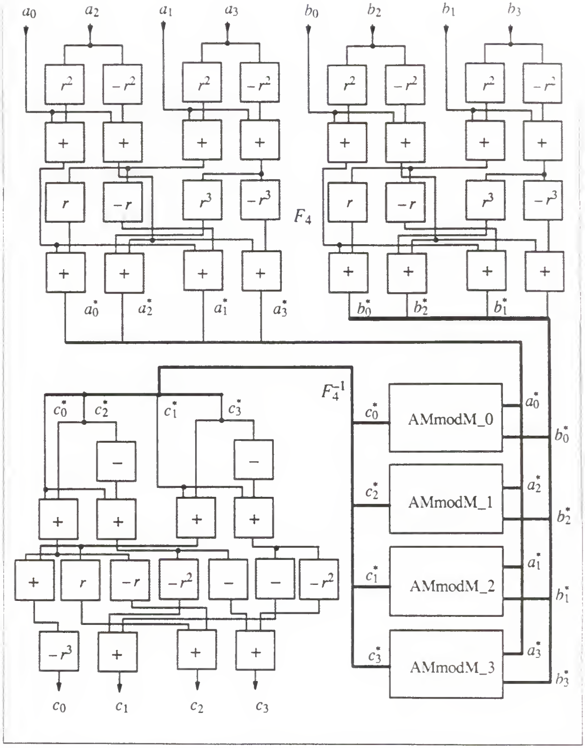


Figure 6.5 System Block of the 9-bit Third-order PRNS Arithmetic Unit with FFT-Type Fast Isomorphic Mappings

between design load and adequacy of test vector generation is made leads to the choice of $m = 8$ which makes $M = 257$. As a result, a 9-bit system is built using the HP Design Capture System (HP-DCS) which provides schematic capture and logic/timing simulations. For the low level design, a λ-based CMOS design is accomplished using the Magic layout tool which runs on a SUN SPARCstation. Since this level of design is for the proof of theory, a 5-bit system is built which contains total of 13,000 transistors on a 6.0 by 6.1 square millimeter footprint. A detailed description on the design, implementations, test and system evaluation of the third-order polynomial residue number processor are presented.

6.3.1 A Nine-bit PRNS Processor Development on the HP-DCS

System Design and Schematic Capture. Applying a top-down hierarchical design approach, a top level system schematic is captured and depicted in Figure 6.6 which consists of both forward/inverse transforms (the forwMAP's and a backMAP) and two processing cores. These cores make up a four-cells modular multiplier banks (the mul9modp's) and a 4-cell modular adder banks (the add9MODp's). A microcode control bit (selma) is used to set the multiplexer arrays to select either multiplication or addition operations. A hierarchical tree graph of the system is shown in Figure 6.7. The HP-DCS support a broad variety device library which includes CMOS, TTL, and analog cells. In terms of function, there are drivers and buffers, flip-flops, decoders, arithmetic, counters, latches and triggers, multiplexers, transceivers, and basic logic gates.

An m-bit third order polynomial RNS processor with the capabilities of manipulating polynomial arithmetic is developed based upon the theories discussed in previous sections. The detailed schematic of logic blocks of the system are demonstrated in the following sections which include the Modular Negator, the Modular Adder, the Modular Multiplier, and the Isomorphic Forward/Inverse Mapping Units. A sequence of the logic and timing diagrams are included to demonstrate the design procedure verify and system performance.

1) Modular Negator

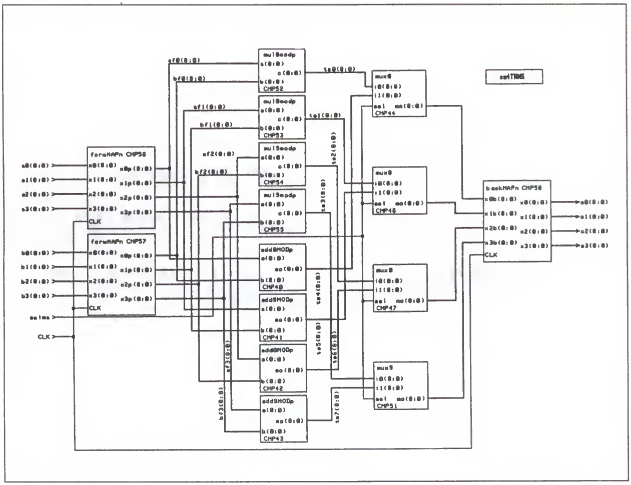


Figure 6.6 Top Level Schematic of the PRNS System

The purpose of the module is to convert an input $x \in \mathbf{Z}_M$ into $-x$, so that it can be used in subtraction. The mapping equation is given by

$$-x = (M - |x|) \bmod M \qquad (6.35)$$

If $M = 2^m + 1$ and $x \in \mathbf{Z}_M$, then

$$\begin{aligned} -x &= (2^m + 1 - x) \bmod M \\ &= (2^m + 1 + \bar{x} + 1) \bmod M \\ &= (2^m + 2 + \bar{x}) \bmod M \end{aligned}$$

where $\bar{x}$ denotes the bit-wise complement (1's complement) of x. From Equation (6.35) shown above, one can find that there exists relations between input x and output $-x$. Given

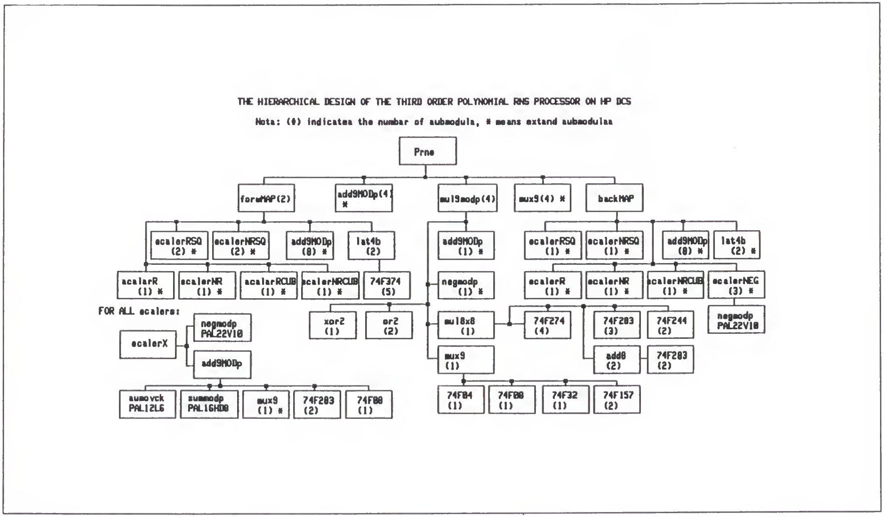


Figure 6.7 The Hierarchical Design of the Third-Order Polynomial RNS Processor on HP DCS

$$x = a_m 2^m + a_{m-1} 2^{m-1} + \ldots + a_1 2 + a_0$$

$$-x = b_m 2^m + b_{m-1} 2^{m-1} + \ldots + b_1 2 + b_0$$

then

$$b_0 = \bar{a}_0$$

$$b_1 = \bar{a}_1$$

$$b_2 = \bar{a}_1 a_2 + a_1 \bar{a}_2$$

$$b_3 = \bar{a}_1 \bar{a}_2 a_3 + (a_1 + a_2) \bar{a}_3$$

$$\vdots$$

$$b_m = \bar{a}_1 \bar{a}_2 \ldots a_m + (a_1 + a_2 + \ldots + a_{m-1}) \bar{a}_m \tag{6.36}$$

For the case of nine-bit system where $m = 8$, a commercial Programmable Array Logic (PAL) is used. The HP-DCS provides a automatic PAL implementation using the Programmable Logic Device Design System (PLDDS) software package. The Foreign Tool Interface (FTI) of the PLDDS is used to perform automatic logic synthesis and device selection according the user defined Boolean equation file. A final PLD design can be transferred to regular DCS design. A file contains Boolean equations for the modular negator is shown in Figure 6.8. A multiwindow picture shown the device selection of FTI is also demonstrated in Figure 6.9. For detail description of the PLD design procedures, see reference in HP PLDDS User Interface Basics.

In order to improve interpretation and reduce the computational delay of negation, the case of input $x = 0$ should be kept at zero rather than mapped to M. Thus, a zero detect circuit is required and embedded in the design.

2) Modular Adder

A modulo M Adder adds two integer inputs in $\mathbf{Z}_M$ and maps their sum into modulo M sum. The same unit can be used to do a modulo M subtraction by using a NEGATOR to negate one of the operands. A Modulo M Adder, shown in Figure 6.10, is actually composed of

```
/* FTI Design for add8modp – negmodp, using PAL22V10 */

dummy main (xi0, xi1, xi2, xi3, xi4, xi5, xi6, xi7, xi8, xo0, xo1, xo2, xo3, xo4,
           xo5, xo6, xo7, xo8)

input      xi0, xi1, xi2, xi3, xi4, xi5, xi6, xi7, xi8;
output     xo0, xo1, xo2, xo3, xo4, xo5, xo6, xo7, xo8;
{
           node     zero0, xo8n;
           zero0    = xi0 | xi1 | xi2 | xi3 | xi4 | xi5 | xi6 | xi7 | xi8;
           xo0      = !xi0 & zero0;
           xo1      =  xi1 & zero0;
           xo2      = (!xi1 & xi2 | xi1 & !xi2) & zero0;
           xo3      = (!xi1 & !xi2 & xi3 | (xi1 | xi2) & !xi3) & zero0;
           xo4      = (!xi1 & !xi2 & !xi3 & xi4 | (xi1 | xi2 | xi3) & !xi4) & zero0;
           xo5      = (!xi1 & !xi2 & !xi3 & !xi4 & xi5 | (xi1 | xi2 | xi3 | xi4) & !xi5)
                      & zero0;
           xo6      = (!xi1 & !xi2 & !xi3 & !xi4 & !xi5 & xi6 | (xi1 | xi2 | xi3 | xi4 |
                      xi5) & !xi6) & zero0;
           xo7      = (!xi1 & !xi2 & !xi3 & !xi4 & !xi5 & !xi6 & xi7  | (xi1 | xi2 |
                      xi3 | xi4 | xi5 | xi6) & !xi7) & zero0;
           xo8n     = (!xi1 & !xi2 & !xi3 & !xi4 & !xi5 & !xi6 & !xi7 & xi8 | (xi1 |
                       xi2 | xi3 | xi4 | xi5 | xi6 | xi7) & !xi8);
           xo8      = !xo8n & zero0;
}
```

Figure 6.8 The Boolean Equation of the Module – negmodp

three parts. One of them is an m-bit fast carry lookahead adder, another is an m-bit table-look-up modulo unit, and the third is a logic control circuit.

a) The m-bit Fast Carry-Lookahead Adder

It can be implemented by conventional full carry-lookahead adder with overflow flag (see Figure 6.10). The combinations of slice-type full adder with carry-generate, carry-propagate outputs and a carry-lookahead generator achieve the fast operating goal.

b) The Table Mapping Modulo Unit (MDL_M)

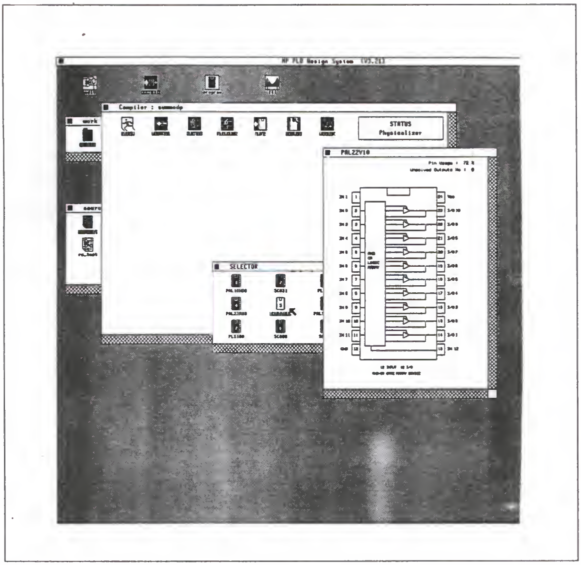


Figure 6.9 The Multiwindow Demonstration of FTI Device Selection

The modulo operation MDL_M is implemented by applying a table mapping modulo unit which uses a programmable logic device to implement a bit parallel output mapping of a binary adder (see Figure 6.11). For thr case that

$$S = A + B \geq M$$

where $M = 2^m + 1$, then

$$(A + B) \bmod M = S - M.$$

The procedure used to realize the unit is

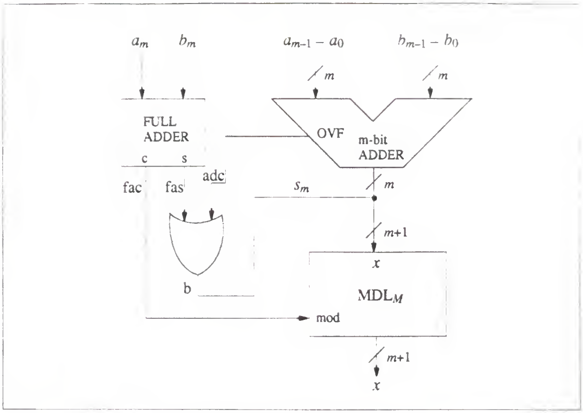


Figure 6.10 The Modulo M Adder

i) Starting with the least significant bit (LSB) of S, complement all '0''s up to the first '1'.

ii) Complement the first encountered '1' in S.

iii) Leave of other bits of S unaltered and set the (m+1)st bit to '0'.

The Boolean equations found from the algorithm shown above are

$$d_0 = \bar{s}_0$$

$$d_1 = \bar{s}_0\,\bar{s}_1 + s_0\,s_1$$

$$d_2 = \bar{s}_0\,\bar{s}_1\,\bar{s}_2 + (s_0 + s_1)\,s_2$$

$$\vdots$$

$$d_{m-1} = \bar{s}_0\,\bar{s}_1\,\bar{s}_2 \ldots \bar{s}_{m-1} + (s_0 + s_1 + \ldots + s_{m-2})\,s_{m-1} \tag{6.37}$$

For the case of $m = 8$, a similar design – summodp – is implemented by the HP-DCS software. The Boolean equation file is shown in Figure 6.12.

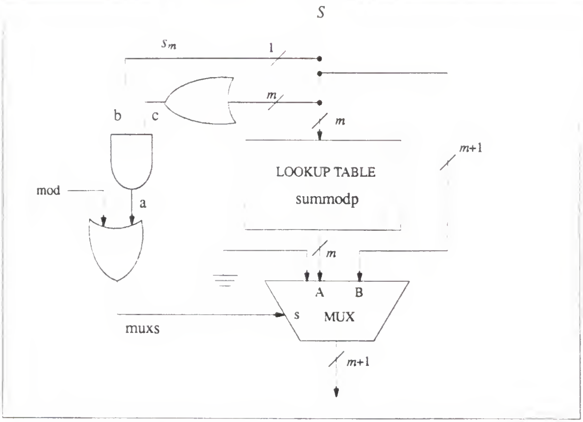


Figure 6.11 The MDL_M Unit Implemented Using a Table Lookup Method

c) Control Logic (Compare Network)

It is simply a combinational logic circuit which decides if the output of the module is to be a moduloed or unmoduloed one. The conditions for modulo are

$$S = A + B \geq M$$

which means

$$\{ \ s_m \ = 1 \text{ and } (s_{m-1} \text{ or } \ s_{m-2} \text{ or} \ldots \text{or } \ s_0 \ ) \neq 0 \ \} \text{ or } \ \{ \text{ mod} = 1 \ \}.$$

A HP-DCS designed PAL is generated by the Boolean equation file which is shown in Figure 6.13 .

Over all, A nine-bit modular adder is implemented in the HP-DCS environment which is demonstrated in Figure 6.14. The usefulness of the programmable logic device is its ability to combine a number of distributed logic gates (the "glue" logics) into a single device. By reprogrammed the device the functionality of the gates is modified to suit specific needs. The

```
/* FTI Design for add8modp – summodp, using PAL16HD8 */

dummy main (xi0, xi1, xi2, xi3, xi4, xi5, xi6, xi7, xo0, xo1, xo2, xo3, xo4, xo5,
          xo6, xo7)

input     xi0, xi1, xi2, xi3, xi4, xi5, xi6, xi7;
output    xo0, xo1, xo2, xo3, xo4, xo5, xo6, xo7;
{
          xo0     = !xi0;
          xo1     = !xi0 & !xi1 | xi0 &  xi1;
          xo2     = !xi0 & !xi1 & !xi2 | (xi0 | xi1) & xi2;
          xo3     = !xi0 & !xi1 & !xi2 & !xi3 | (xi0 | xi1 | xi2) & xi3;
          xo4     = !xi0 & !xi1 & !xi2 & !xi3 & !xi4 | (xi0 | xi1 | xi2 | xi3) & xi4;
          xo5     = !xi0 & !xi1 & !xi2 & !xi3 & !xi4 & !xi5 | (xi0 | xi1 | xi2 | xi3 |
                    xi4) & xi5;
          xo6     = !xi0 & !xi1 & !xi2 & !xi3 & !xi4 & !xi5 & !xi6 | (xi0 | xi1 |
                    xi2 | xi3 | xi4 | xi5) & xi6;
          xo7     = !xi0 & !xi1 & !xi2 & !xi3 & !xi4 & !xi5 & !xi6 & !xi7 | (xi0 |
                    xi1 | xi2 | xi3 | xi4 | xi5 | xi6) & xi7;
}

*Note : & = AND; | = OR; ^ = XOR; ! = INV.
```

Figure 6.12 The Boolean Equation of the Module – summodp

modular adder consists of two TTL four-bit adders (74283's), two PALs (the summodp which is the table mapping modulo unit whose Boolean equation file is shown in Figure 6.12 ; and the sumovck which accomplishes the compare network task whose Boolean equation file is shown in Figure 6.13), and a nine-bit multiplexer. The multiplexer module is made up by two four-bit multiplexer (74157's) and gates which include inverter (7404), two-input AND gate's (7400's), and two-input OR gate (7432). The detail circuitry can be found in the reference.

3) Modular Multiplier

The single modulus RNS multiplier is a critical element to the successful realization of the PRNS processor. Using multiplier devices which are available on the market, the Modulo

```
/* FTI Design for add8modp – sum overflow checking */
/* Module sumovck using PAL12L6 */

dummy main (a8, b8, adc, d0, d1, d2, d3, d4, d5, d6, d7, muxs, o8)

input     a8, b8, adc, d0, d1, d2, d3, d4, d5, d6, d7;
output    muxs, o8;
{
          node    a, b, c, fac, fas;

          muxs    = or(fac, a);
                  a     = and(b, c);
                  b     = or(fas, adc);
                  c     = d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7;
                  fac   = a8 & b8;
                  fas   = a8 ^ b8;
          o8      = or(fas, adc);
}
*Note : & = AND; | = OR; ^ = XOR; ! = INV.
```

Figure 6.13 The Boolean Equation of the Module – sumovck

M Multiplier can be realized in a very straight forward manner. There are some special cases which should be taken into account separately (see figure 6.15).

i) If both inputs A and B are equal to 2^m then C (the result) equals 1, since (2^m 2^m) mod M = 1 for $M = 2^m + 1$. The detect circuit will enable the output buffer to set the result to be 1.

ii) If only one of the input equals to 2^m (for example $A = 2^m$), then

$$\begin{aligned}(A\,B) \bmod M &= (2^m B) \bmod M \\ &= ((2^m + 1)B - B) \bmod M \\ &= -B.\end{aligned}$$

The detect circuit will enable the output buffer to set the result to be the negated B.

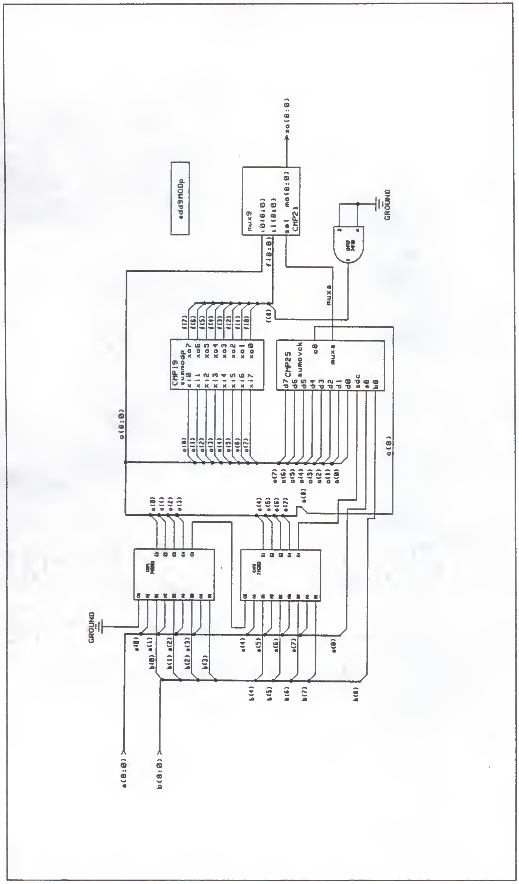


Figure 6.14 A Nine-Bit Modular Adder

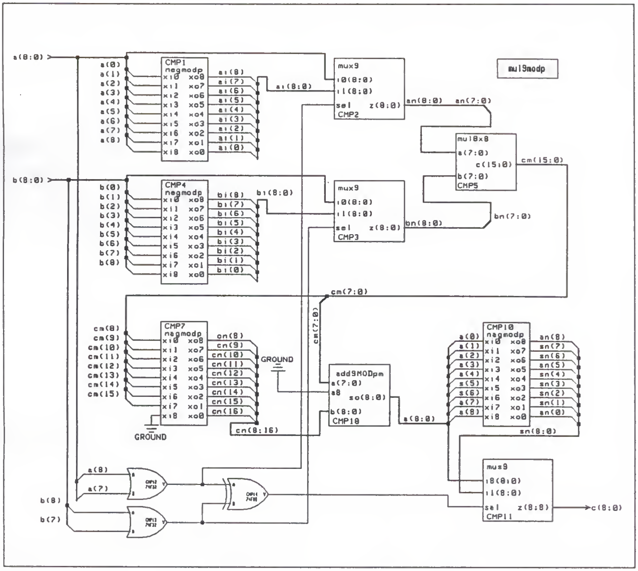


Figure 6.15 The Modulo *M* Multiplier

These special cases are solved by using a shortcut through the Negators and some elementary gates, instead of going through the lengthy data path of all processing units.

If the (MSB–1)th bit is 1, then this indicates the operand is negative (except for the 2^{m-1} case) and shall be negated before input to the multiplier. The unsigned product shall be moduloed and negated if required. For the modulo operation, the unsigned multiplier can be configured using the following split-field scheme. Consider the full precision unsigned product M_u ($\leq 2m$ bits), partitioned as

$$M_u = 2^m P_H + P_L$$

where P_H and P_L are the high part and low part of the intermediate product M_u. This yields a modulo M value M_d

$$M_d = M_u \bmod (2^m + 1) = P_L - P_H .$$

The final result will be ($P_L - P_H$) or the negation of ($P_L - P_H$) depending on the polarity of inputs. From the scheme shown in Figure 6.15, the longest propagation delay is associated with the unsigned multiplication of positive and negative operands. The delay is essentially that of the three Negators, one Modulo M Adder, some multiplexers, and an unsigned multiplication. Note that the unsigned eight by eight multiplier (mul8×8) is implemented by four four-bit multipliers (74274's) and a couples of four-bit adders. A detail circuitry is shown in Appendix B.1.

4) Isomorphic Forward/Inverse Mapping Units

These units are designed to perform the the isomorphic mappings between polynomial domain (or the algebraic integer domain for the complex number approximation). As the discussions in the previous sections, the mappings are expedited using the FFT-type algorithms. However, the fundamental operations still require the scalings by some forms of the root r of the equation $x^N + 1 = 0$. These scalar modules, as shown in Figure 6.3, include $\pm r$, $\pm r^2$, and $\pm r^3$. With a special treatment, the multiplier-free scalings are achieved by simply transform tasks of addition or subtraction. A sample of one of the scalar modular, $r^2 x$, designed in the HP-DCS package is depicted in Figure 6.16. Other scalar modules can be found in Appendix.

Using these scaling elements, the isomorphic mappings are implemented in a two-stage pipeline structure based on the FFT-type algorithm. The forward mapping and the inverse mapping are shown in Figure 6.17 and Figure 6.18, respectively. The latch element, lat4b, which is designed to store the intermediate result of the pipeline is composed of a series of TTL latch devices (74374's). Its detail design circuitry can also be found in Appendix B.1.

Functional Verification and Timing Analysis. The most important and crucial task of the system development is to verify the functionality and the performance of the design. The

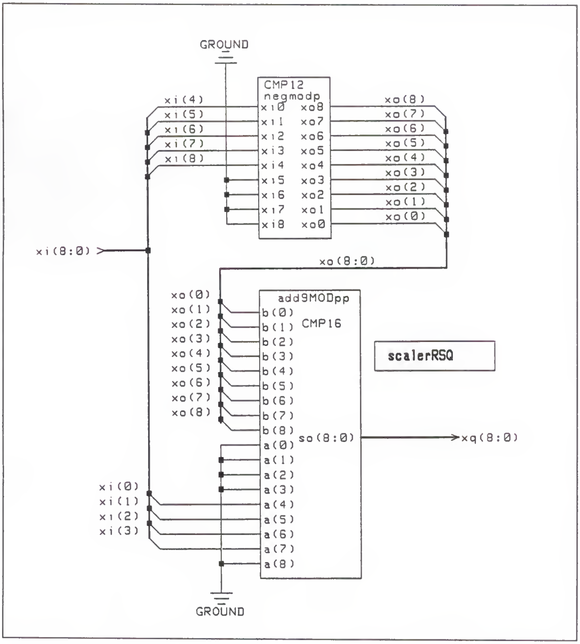


Figure 6.16 The Scalar Module RSQ

HP-DCS supports both logic and timing simulation by the DVI software package (the DVI stands for HP Design Verification Interface for HILO-3) which is to simulate circuits created by the HP-DCS.

The basics of the DVI includes: a) Waveform Display shows how to use simulation features to examine waveforms; b) Simulation Errors covers simulation failure, debugging, and re-simulation; c) Min Time Simulation deals with minimum time delay simulation parame-

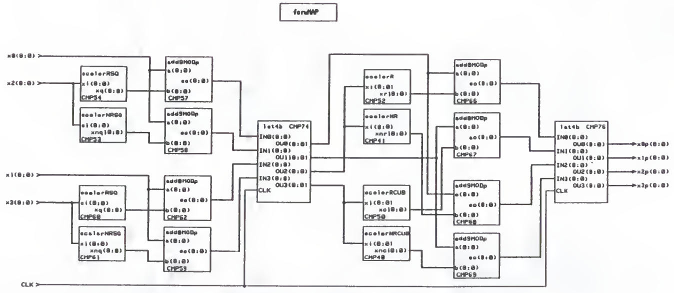


Figure 6.17 The Isomorphic Forward Mapping Module

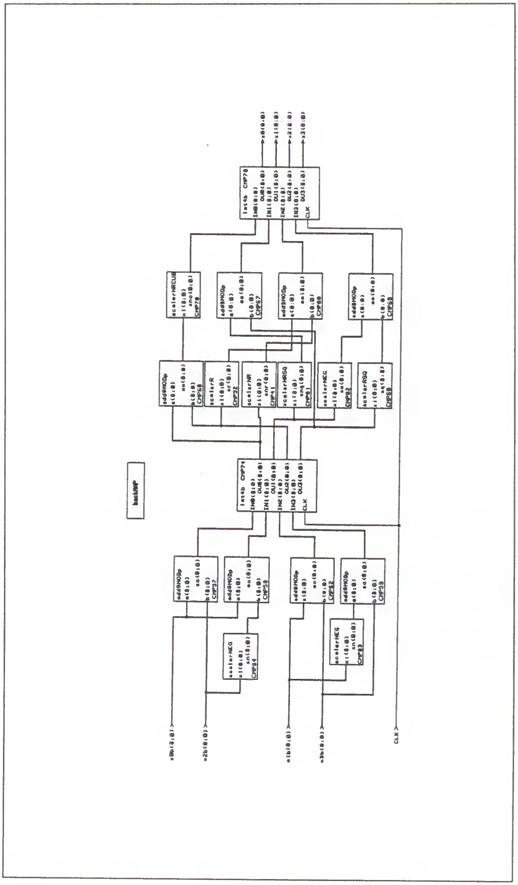


Figure 6.18 The Isomorphic Inverse Mapping Module

ters and how to compare simulations with differing parameters; d) Functional Modeling introduces modeling through the use of the HP-DVI model template; and e) Memory Models and Mapping Files discusses memory model building and its use in simulation.

An illustration shown the logic and timing information of the nine-bit third-order PRNS processor is demonstrated in Figure 6.19. This diagram shows the hexadecimal repre-

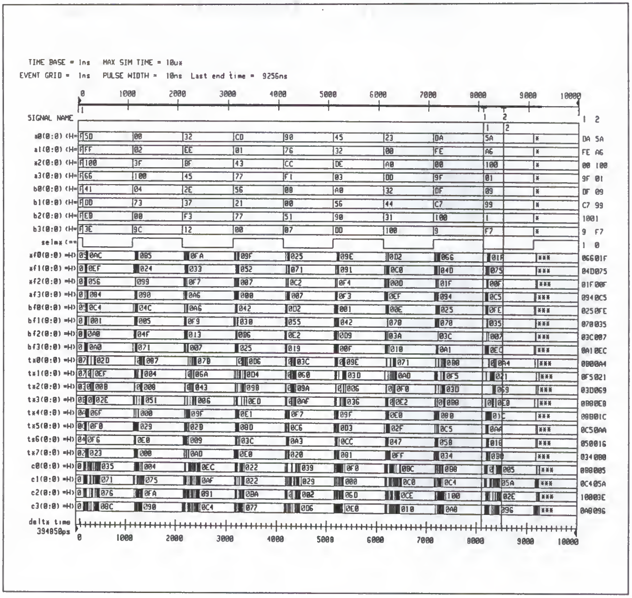


Figure 6.19 The Logic and Timing Simulation Result of the Third-Order PRNS Processor

sentation of the results along the processing pipeline which are indicated by various signal names, such as a0 – a3, b0 – b3, c0 – c3, af0 – af3, ts1 – ts7 etc. The horizontal axis indicates the timing frame which represents the time elapse needed for an input stimulus to reach a certain point of the pipeline. In this diagram, the time base equals to 1 nano second. The total package of the simulation for each of the elements of the processor is attached in Appendix B.2. A table which summarizes the timing information of the processor is shown in Table 6.1 where the highlight item shows the propagation delay of the PRNS processor.

Table 6.1 The Timing Information of the Simulation of the PRNS Processor

Module \ Delay Time	min.	max.	Module \ Delay Time	min.	max.
lat4b	6.95	7.05	add9modp	45.45	45.85
mux9	8.99	9.05	mul8x8	71.25	71.88
negmodp	17.52	17.91	forwmapn	88.96	89.12
summodp	23.09	23.12	forwmapn (clk)	178.96	189.12
sumovck	24.69	24.98	backmapn	93.06	93.53
scalerNR	34.07	34.13	backmapn (clk)	201.96	213.12
scalerNRSQ	34.02	34.13	mul9modp	172.06	172.91
scalerNRCUB	39.92	40.75			
scalerRCUB	55.82	55.97	sm4TRNS	367.06	378.03
scalerRSQ	56.82	56.96			
scalerR	63.75	63.88			
*Note: Time Base = ns					

6.3.2 A Five-bit PRNS Processor Development on the Magic CAD Tool

A transistor level implementation of the third order modulo 17 polynomial RNS arithmetic unit is done on the MAGIC CAD tool in a SUN SPARCstation. By applying 2-μ m single-poly double-metal CMOS technology, this unit consists of 13,002 transistors and has footprint of 6,150 mm by 6,050 mm. Two transistor-level designs within the development of the PRNS processor which include a 5-bit modular multiplier as well as a complete 5-bit polynomial RNS arithmetic system. In this hierarchical top-down design approach, a large number of the subcells which are used to build the modular multiplier can also be used in the design of the PRNS processor. Even the modular multiplier itself becomes a major subcell of the PRNS processor.

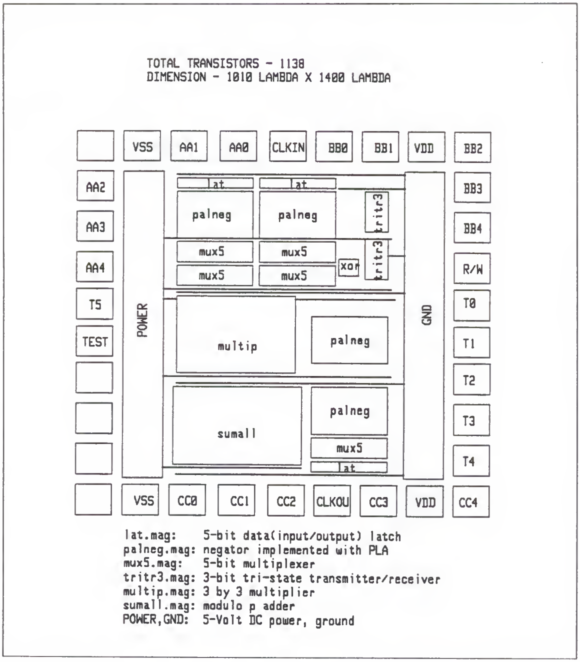
The Five-Bit Modular Multiplier.

1) Floorplan

By applying the design database created in the HP-DCS design, the schematics of the modular multiplier in Figure 6.15 is developed. Consequently, the block placement and wiring diagram of the modular multiplier is shown in Figure 6.20 where the design consumes 1138 transistors (include both n-type and p-type transistors) with a dimension of 1010λ by 1400λ and has 29 I/O connections.

The system blocks includes: 1) lat.mag (latches); 2) palneg.mag (PLA version of negators); 3) mux5.mag (5-bit 2-to-1 multiplexer); 4) tritr3.mag (tri-state 3-bit transmitter/receiver); 5) multip.mag (3 by 3 multiplier); 6) sumall.mag (modulo M adder); and 7) padin.mag, padout.mag, padgnd.mag, padvdd.mag (I/O pads and power/ground pads) and internal power/ground rails.

The Input/output signals of the system are: 1)AA0–AA4 (5-bit inputs); 2)BB0–BB4 (5-bit inputs); 3)CC0–CC4(5-bit outputs); 4)T0–T5 (Test bus); 5)VSS (Ground); 6)VDD (Power); 7)CLKIN (Clock-in); 8)CLKOU (Clock-out); 9)R/W (Test bus read/write);



```
TOTAL TRANSISTORS - 1138
DIMENSION - 1010 LAMBDA X 1400 LAMBDA
```

lat.mag:	5-bit data(input/output) latch
palneg.mag:	negator implemented with PLA
mux5.mag:	5-bit multiplexer
tritr3.mag:	3-bit tri-state transmitter/receiver
multip.mag:	3 by 3 multiplier
sumall.mag:	modulo p adder
POWER,GND:	5-Volt DC power, ground

Figure 6.20 Floorplan of the Five-bit Modular Multiplier

10)TEST (Test mode enable). The data flow of the design is from top to bottom and local power/GND rails to each cell are horizontally routed.

A tree graph shown in Figure 6.21 summarized hierarchically the components required to build the modular multiplier. The top-bottom tree shows the components used in different levels of the design and also indicates the number of cells used in each module. A list of mag-

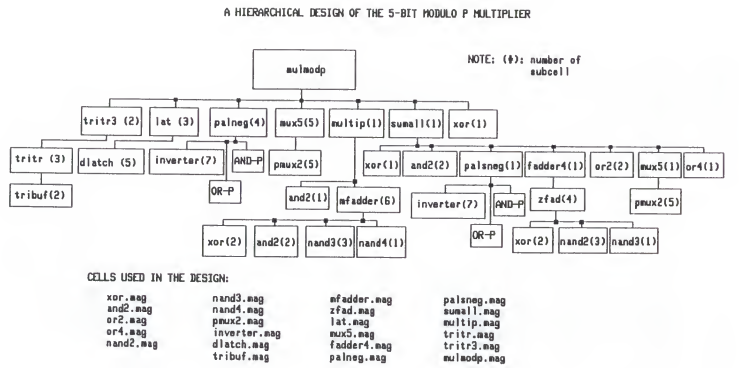


Figure 6.21 A Hierarchical Design of the Five-Bit Modular Multiplier

ie file name of cells is also included. Any bigger cell can be built by "stacking" a number of subcells which is called the primitive building cells such as xor.mag (exclusive-OR Magic cell), nand2.mag (Two-input NAND Magic cell) etc. For example, a cell, tritr3.mag (3-input tri-state trans/receiver), is made of three tritr's (single-bit tri-state trans/receiver) which is also made of two tribuf's (single-bit tri-state buffer).

2) Logic Building Blocks

The logic building blocks of the system are designed based upon the HP-DCS designs. Most of the detailed Magic layouts are demonstrated in Appendix B.3. A brief listing of these elements are shown below along with the description of some basic design methodology of gates such as inverter (inv), NANDx (e.g. nand2). I/O pads (e.g. inpad) and the Programmable Logic Array (PLA) device.

The Complementary MOS (CMOS) circuit (structural) designs of an inverter and a two-input NAND gate is shown in Figure 6.22. For the inverter cell, when there is a '0' on the

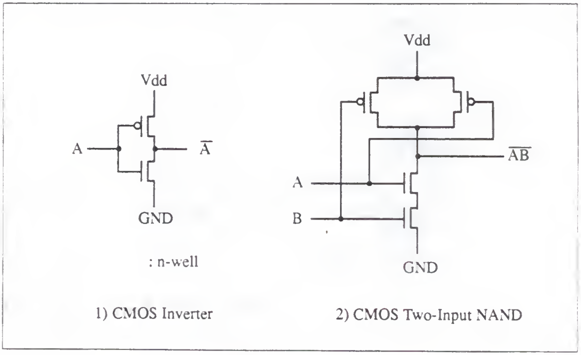


Figure 6.22 Basic CMOS Cells

input, there is a '1' at the output. In general, the lower transistor (n-type) only has to pass a '0' while the upper transistor (p-type) has to pass a '1'. At any moment (except the short transient period) the current never flow from Vdd to GND. This power saving feature is the primal advantage of CMOS technology. For the NAND gate, the '0' propagates to the output only when both A and B turn on the lower n-type transistors. A Postscript (see Appendix B.4.2) printout of a Magic layout cell, nand2, is demonstrated in Figure 6.23.

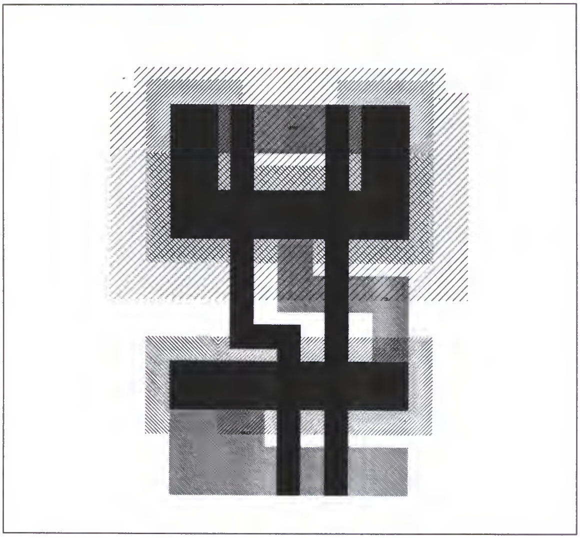


Figure 6.23 A Two-Input CMOS NAND Gate Magic Layout

Another important device used in the design is PLA where a typical PLA uses an AND-OR structure similar to that shown in Figure 6.24. This implementation also shows clocks to latch inputs and outputs. The basis for a PLA is sum of products form of representation of binary expressions. The most straight-forward PLA design uses a pseudo-nMOS NOR gate

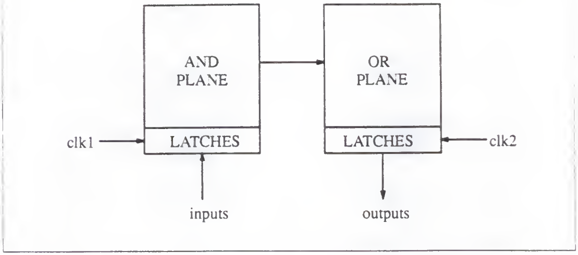


Figure 6.24 An AND-OR Programmable Logic Array

which has the advantages of simplicity and small size. However, its disadvantage occur due to the static power dissipation of the NOR gates.

The circuit diagram of an example PLA (Figure 6.25) will help to clarify the structure and function of the AND-OR planes of the PLA.The input register bit for each input path is formed by a pass transistor clocked on $\underline{clk1}$ leading to both inverting and noninverting super buffers. The outputs of the AND plane are formed by horizontal lines with pull-up transistors (whimppy pull-up) at their leftmost end. The function of the PLA's AND plane is then determined by the locations and gate connections of pull-down transistors connecting the horizontal lines to ground. Each output running horizontally from the AND plane carries the NOR combination of all input signals that lead to the gates of transistors attached to it. For example, the horizontal row labeled R3 has three transistors attached to it in the AND plane (that is $A + \overline{B} + C$). If any of these inputs is high, then R3 will be pulled down toward ground. Thus, R3 is the NOR of $(A + \overline{B} + C)$ which is $R3 = \overline{A + \overline{B} + C}$.

Once again, each of the outputs of OR plane is the NOR of the signals leading to the gates of all transistors attached to it. For example, both R1 and R3 lead to the gates of transistors leading from the output line $\overline{z2}$ to ground. Thus, z2 = INV(NOR(R1, R3)) = OR(R1, R3)

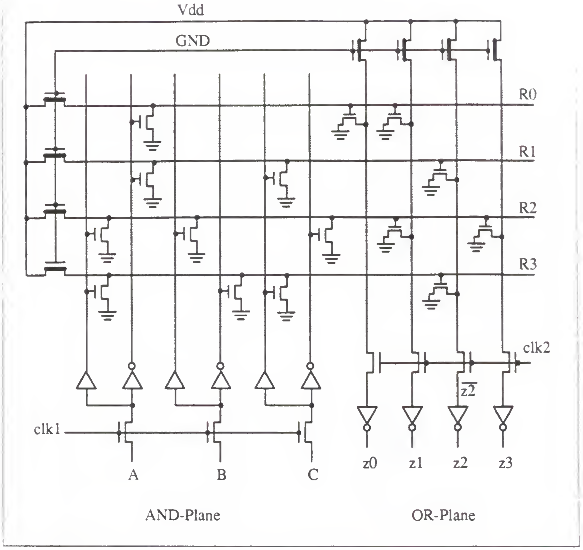


Figure 6.25 A Pseudo-nMOS Programmable Logic Array Design

$= R1 + R3 = (\overline{\overline{A} + C}) + (\overline{A + \overline{B} + C}) = A\overline{C} + \overline{A}\,B\overline{C}$, which appears directly as the sum of products canonical form of Boolean functions of the PLA inputs, as the OR of AND terms.

Of all the CMOS circuit structures, Input/Output (I/O) structures require the most amount of circuit design expertise in association with detailed process knowledge. It is often convenient to build I/O pads with a constant height and width, with connection points at pre-specified locations. Pad size is defined usually by the minimum size to which a bond wire can be attached. Other design considerations which include the sufficient drive capability of output pad, the susceptibility to latch-up problem, and the reliability problems related to oxide breakdown due to the electrical overstress (EOS) or electrostatic discharge (ESD) phenomenon. Usually a combination of a resistance and diode clamps are used to limit this potentially destructive voltage. A typical input pad circuit is shown in Figure 6.26. Clamp diodes D1 and

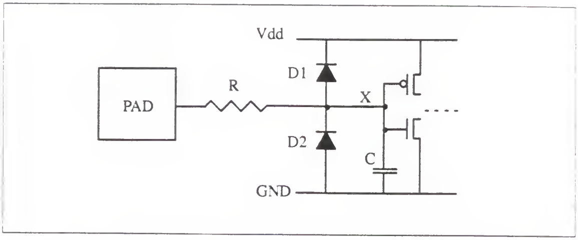


Figure 6.26 Typical Input Protection Circuit

D2 turn on if the voltage at node X rises above Vdd or below GND. resistor R is used to limit the peak current that flows in the diodes in the event of an unusual voltage excursion.

The following elements are the building block of the five-bit modular multiplier shown in Figure 6.20. The detailed cell placement which support the total number of transistor used and the size of the cell for each element in the design hierarchy can be found in Appendix B.3.

a) mfadder — This cell is a modified full adder which has four bits input and a carry-in bit. The design consists of two xor.mag's, two and2.mag's, three nand3.mag's, and a nand4.mag. The cell is a special design which is dedicated to the implementation of the parallel multiplier cell, multip;

b) multip — The three by three multiplier which consists of six mfadder cells is capable of performing fast parallel multiplication;

c) zfad — This is a regular 2-bit input, 1-bit carry-in full adder which is a subcell for implementing the four-bit adder, fadder4;

d) fadder4 — A four-bit adder;

e) sumall — This is a modular adder which consists of a four-bit full adder cell, fadder4, a PLA version of modulo converter palsneg, glue logic gates of sum overflow checker, and a five-bit multiplexer mux5. The task of the module is to perform addition with the capability of overflow checking. In the case of overflow, the modulo converter palsneg converts the result to conform within the of range *M*;

f) palneg — A PLA implementation of the Boolean equations of Figure 6.8 performs the negation task and its transistor layout is demonstrated as an example in Figure 6.27;

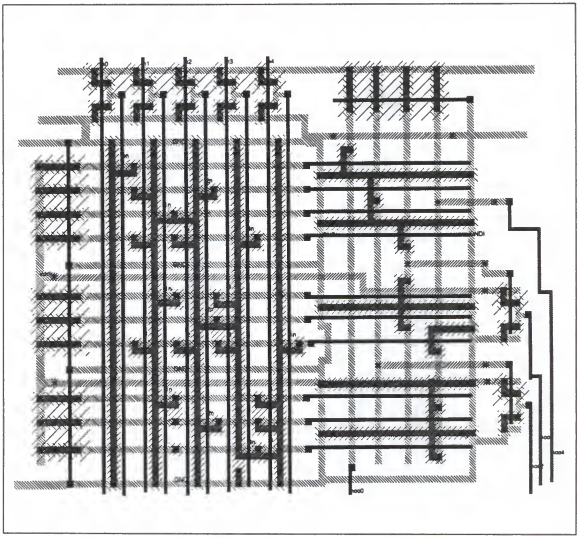


Figure 6.27 The PLA Transistor-Level Layout of the Negator Cell – palneg

g) plasneg — A PLA implementation of the Boolean equations shown in Figure 6.12 which performs the sum modulo converting;

h) lat — A five-bit latch. The latches lat give the system the capability of synchronized operation with outside world and reduce the skew of signal flow in pipeline processing;
i) mux5 — A five-bit multiplexer. The 2-to-1 5-bit multiplexer mux5 is part of the decision-making circuits which are used in many occasions in the design. Although the bits of mux5 are not fully used in some places, the compactness and modularity of the cell is shown to be handy in hierarchical design, such that they compensates the waste in bits; and
j) tritr3 — Three-bit tri-state transmitter/receiver buffer. The tri-state buffer tritr3 is a bidirectional transmitter/receiver circuit with read/write controls. The module is used as embedded test circuit that make the system observable and controllable..

The Negator and the sum modulo converter are implemented by the Programmable Logic Array (PLA) subcells planeg and palsneg respectively. All these modules have a regular structure and require high speed processing capability. The pseudo nMOS PLA is a good candidate for this application. A CMOS design of the modular multiplier with I/O pads is demonstrated in Figure 6.28. All of its subcells with the Magic layout are collected in the Appendix B.3.

3) Testability Plan

A trade-off is made to the system testability and its hardware budget. The built-in test module is implemented by using transmitter/receiver tritr3s and a pair of multiplexer mux5 (see Figure 6.29)

a) What to test

The test point is located at the core of the system multip which is at the midstream of the data flow. If there is any processing errors exist, the errors are detected immediately and the location of these errors can be determined wherever at upstream or downstream. It would be helpful to have test points at each module along the data flow. However, disadvantages show up due to more hardware required and gate delay added to data propagation.

b) How to test

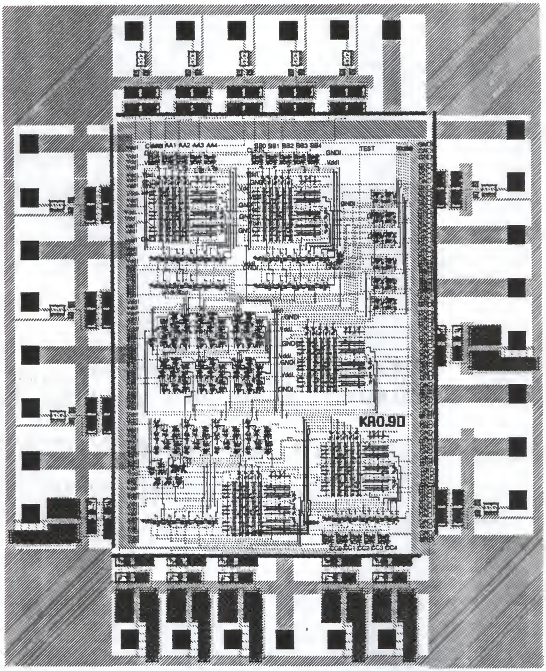


Figure 6.28 A CMOS Magic Layout of the Five-bit Modular Multiplier

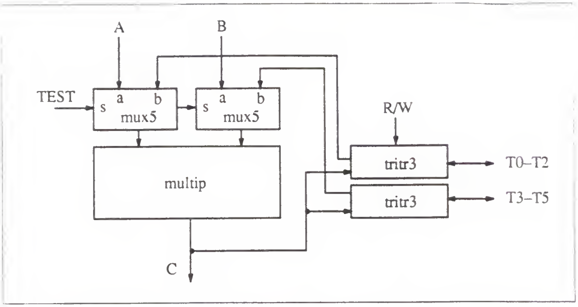


Figure 6.29 The Embedded Test Circuitry of the Five-bit Modular Multiplier

The test circuits can be operated in either normal operation monitoring mode or test mode. In normal operation, Observability is achieved by monitoring (read data out of) the multiplier multip. To perform the monitoring, the signal TEST is Grounded to make selection of A and B as the input of the mux5. READ signal is Inserted to make tritr3s to serve as transmitters such that the output data of multiplier, multip, can be observed. In the case of TEST mode, Controllability is achieved by forcing known states (e.g. inputs) to the multiplier multip and read its output to compare with the precomputed results. To perform the testing, the signal TEST is set to high (TEST enable) such that the normal inputs A and B are blocked. The inputs to the multiplier now are from the testing data T0 – T5 by the control of R/W signal which set the tritr3s as receivers. The data to tested are latched to the tritr3s after a period of time which is the processing delay of the multiplier. The timing diagram for the normal operations and testing is shown in Figure 6.30 which includes three types of timing:

c) Normal operation mode

The two-phase clocks CLKIN and CLKOU are used to strobe inputs and outputs of the modular multiplier. The phase between the clocks should be arranged to ensure that the results are strobed at the right clock edge. This means that the period of the difference in phase

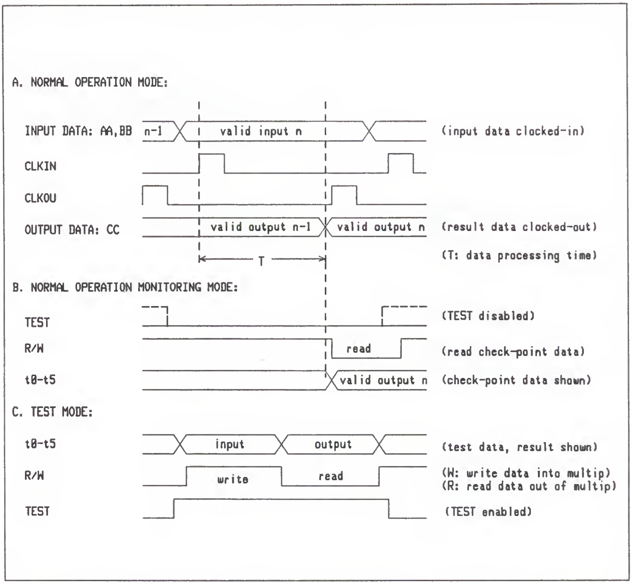


Figure 6.30 The Timing Diagram of the Operation of the Modular Multiplier

should be larger than data processing time T of the multiplier to ensure correct results be obtained.

d) Normal operation monitoring mode

In the normal operation mode, R/W (read/write) signal stays low when the CLKOU is high to monitor the valid data of the multiplier. The signal TEST always stays low in this mode.

e) Test mode

The 'observe' the operation of the multiplier, the signal TEST stays high (TEST enable). The R/W signal is to set the direction of the tritr3s which complies with the 'input' of the testing data and the 'output' of the result.

The Five-Bit Third-Order Polynomial RNS Processor

1) Floorplan

Similar to the procedure of the development of the modular multiplier, the block placement and wiring diagram of the polynomial RNS processor is layouted and shown in Figure 6.31. The total design uses 13,002 transistors with dimension of 6150λ by 6050λ and has 72 I/O connections.

The system blocks includes: 1) lat.mag (latches); 2) fmap.mag (the isomorphic forward mapping) which includes sumall.mags (the modular adder), lat.mag, and a number scalars such as scalerR.mag, scalerNR.mag, scalerRSQ.mag, scalerNRSQ.mag, scalerRCUB.mag, scalerNRCUB.mag; 3) bmap.mag (the isomorphic inverse mapping) which includes scalerNEG.mag in addition to the subcells which are used in the fmap.mag; 4) mulmodp.mag (the modular multiplier); and 5) padin.mag, padout.mag, padgnd.mag, padvdd.mag (I/O pads and power/ground pads) and internal power/ground rails.

The Input/output signals of the system are: 1)A0–A19 (5-bit per tupple, four tupples); 2)B0–B19 (the same as inputs A); 3)C0–C19 (the same as inputs A); 4)Vss (Ground); 5)Vdd (Power); 6)CLK1 (phase 1 clock); 7)CLK2 (phase 2 clock). The data flow of the design is from top to bottom and the local power/GND rails to each cell are horizontally routed.

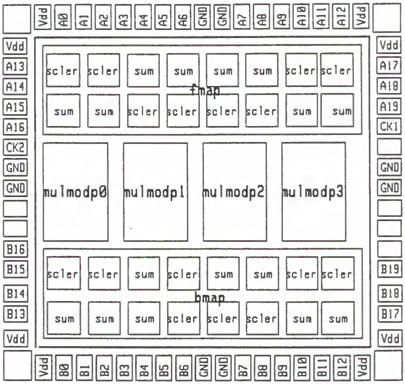
A tree graph shown in Figure 6.32 summarized hierarchically the cells required to build the polynomial RNS processor. Notice that all the basic cells used in the design are the same cells used in the design of the five-bit modular multiplier.

2) Logic Building Blocks

a) Scaler modules include scalerNEG.mag, scalerR.mag, scalerNR.mag, scalerRSQ.mag, scalerNRSQ.mag, scalerRCUB.mag, and scalerNRCUB.mag. These cells are the basic

THE FLOORPLAN OF THE 5-BIT POLYNOMIAL RNS PROCESSOR

TOTAL TRANSISTORS : 13,002
DIMENSION : 6150 um X 6050 um
TECHNOLOGY : 2-um Double-Level Metal CMOS



Vdd, GND: 5-Volt DC power, Ground
Ax: Inputs
Bx: Outputs
CK1, CK2: Two-phase system clocks
fmap: Isomorphic forward mapping
bmap: Isomorphic backward mapping
scler: Mapping scaler, submodule of fmap and bmap
sum: Modular adder, submodule of fmap and bmap
mulmodpx: Modular multipliers

Figure 6.31 Floorplan of the Five-Bit Polynomial RNS Processor

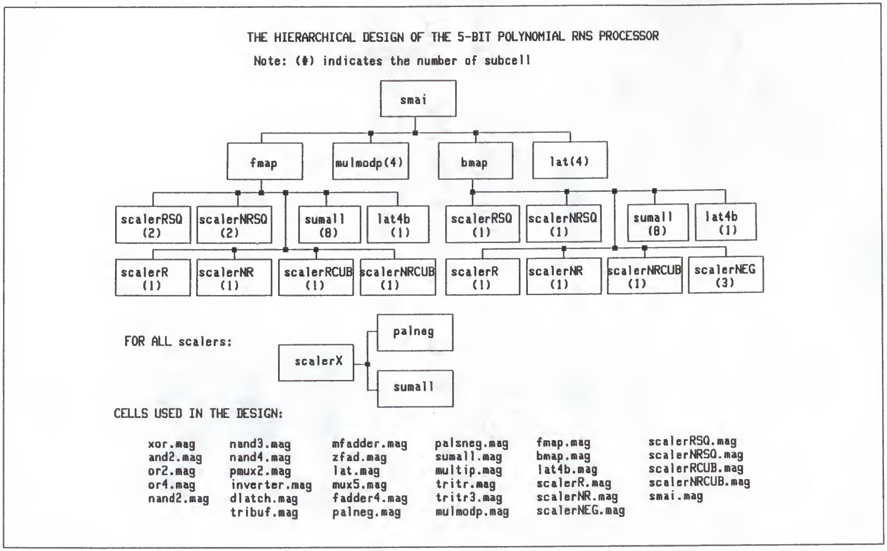


Figure 6.32 A Hierarchical Design of the Five-Bit Polynomial RNS Processor

building blocks of the isomorphic forward/inverse mappings. These layouts are the physical implementation of the block diagram shown in Figure 6.3. These cells are architected with a palneg.mag (Negator) and a sumall (Modular adder) which is a common structure of all the scaler modules as shown in Figure 6.33. Figure6.34 shows an example of a physical layout

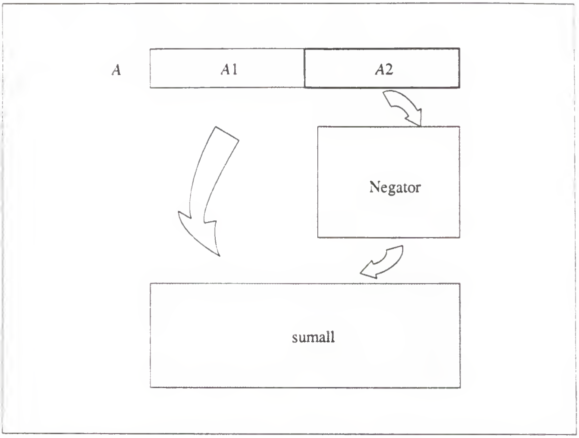


Figure 6.33 The Split-then-Add Scaler Module

of the scaler cell, scalerRSQ.mag;

b.2) fmap.mag — This cell is designed according to the FFT-type transform and the multiplier-free scaling scheme discussed in previous sections and performs the isomorphic forward mapping;

b.3) bmap.mag — This cell performs the isomorphic inverse mapping which has similar structure as that of fmap.mag;

b.4) mulmodp — The cell which is developed in preceding section performs the processing task (modulo multiplication); and

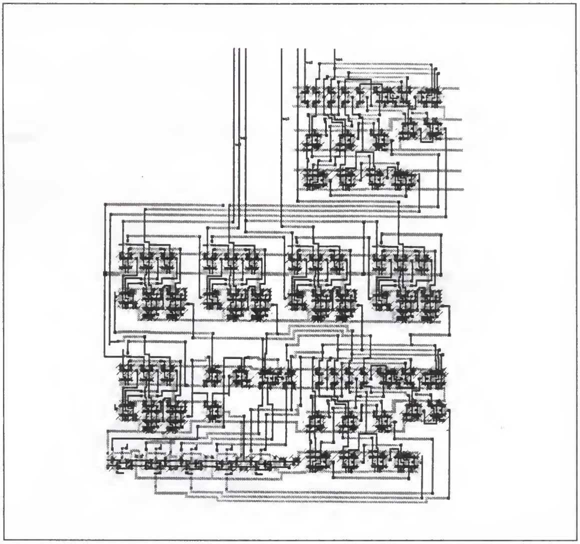


Figure 6.34 A Transistor Layout of the Scaler Module scalerRSQ

b.5) lat — A five-bit latch. The latches lat give the system the capability of synchronized operation with the outside world while reducing the skew of signal flow in pipeline processing.

The Magic Design Rule Summary.

1) Signal routing fashions: a) power/ground rails (fingers) route through each cell across top and bottom horizontally via metal1; and b) signals route through cells vertically via polysilicon layers. When signals need to have horizontal routes, polysilicon layers are contacted

with metal1 which can be placed horizontally. Once the destination is reached, metal1 contacts the polysilicon which continues the routing (see the cell – mfadder);

2) Since the power and ground sources are placed horizontally on the top and the bottom of each cell, every other cell is flipped vertically to reduce the number of horizontal fingers when the cells are stacked together (see the cell – mfadder);

3) Symmetrical cell layout makes it more efficient to pack cells together by simply flipping and rotating the cells (see the cells – zfad or mfadder);

4) To improve the current drain, let the thick power/ground rails locate at the chip boundary and make the fingered rails go across to each cell (see the cell – mulmodp);

5) Labeling each signal (include power/ground rails) allows for a much more efficient routing and tracing (see the cell – mulmodp);

6) A rule of thumb is to avoid applying too much metal2 routing in the lower level (local) design. A clean low level design will make metal2 route to anywhere without having difficulties when routing at the top of the hierarchy (see the cell – mulmodp);

7) In the PLA design, ground rails are placed in every three or four gates in both AND- and OR planes. This compensates the slow propagation inherited from n+ diffusion strips (see the layouts of the PLA cells – palneg or palsneg);

8) To avoid latchup problems in the CMOS design, each cell is embedded with improved substrate contacts. ESD protected guard rings also are included in I/O pads;

9) To apply full advantage of various CMOS design methods to the system development, there are a number of technologies utilized which include static CMOS gates (combinatorial and sequential), pseudo-nMOS gates, pass transistors, tri-state logic and programmable logic arrays (PLAs).

CHAPTER 7
CONCLUSION AND FUTURE RESEARCH

7.1 Conclusion

This dissertation examines some properties of the finite computational structures such as finite groups, rings, and fields, and extends its applications to develop fast algorithms and to build high-speed low-complexity very large scale integrated circuit (VLSI) systems for digital signal processing. A finite polynomial ring structure is addressed to expedite and simplify the multiplication of polynomials. The applications of this polynomial structure are in the areas of short-block length cyclic convolution and complex number arithmetic which originates from the algebraic integer approximation concept. A new algorithm of finite field transforms with cyclic convolution property (CCP) is investigated. This algorithm combines abstract algebraic concepts which include normal basis representation and conjugacy property with factor-type fast Fourier transform algorithms (FFT) to expedite finite field transforms.

Applications in digital filtering, correlation studies, radar matched filtering, and the multiplication of very large integers are based on digital convolution, which can be implemented most efficiently by the NTT method with some constraints. The arithmetic required to accomplish the NTT is exact and involves additions, subtractions, and bit shifts. As in the case of DFT, fast algorithms also exist for the NTT. These transforms are defined on finite fields and rings of integers with all arithmetic performed modulo an integer. The family of NTT includes Fermat, Mersenne, Rader, pseudo-Fermat, pseudo-Mersenne, complex Mersenne, and complex Fermat transforms. They are truly digital transforms and their implementation involves no round-off error. NTT implementation requires additions, subtractions, and bit shifting, but usually does not require multiplications. Others possess fast algorithmic structures similar to those of the FFT.

The trend of VLSI system design utilizes high levels of integration based on efficient algorithms. To construct these algorithms, one must be familiar with the powerful structures of number theory and modern algebra. Since the structures containing the set of integers, polynomial rings, and finite fields play an important role in the design of signal processing algorithms, Chapter 2 investigates the fundamental properties of these finite computational structures. The discussion begins by exploring the concepts of congruence and residue reduction. Discussion of the Chinese remainder theorem in relation to both integers and polynomials leads to the realization that the RNS is capable of carry-free high speed DSP processing and a better understanding of the relationship between the polynomial algebra and digital convolution.

The parallel between the ring of integers and the ring of polynomials over a field is apparent. Both are special cases of an algebraic structure. The field of integers modulo a prime number is the most familiar example of a finite field, but many of its properties extend to arbitrary finite fields, such as GF(p^m). Since the finite field GF(p^m) can be regarded as a vector space of dimension m over GF(p), the basis representations of a field element in an extension field GF(p^m) over its ground field GF(p) are worth investigating. Due to their respective distinct features which make them suitable for specific applications, this research effort investigates three basis types: primal basis, normal basis, and dual basis. The normal basis system is effective in performing operations such as finding inverse elements and power forming which leads to useful applications in finite field transforms.

In the process of constructing GF(p^m) from the ground field GF(p), both the power form and the polynomial form for the nonzero elements of GF(p^m) were developed. In Chapter 3, the structures and applications of these representations are investigated according to the usefulness in various fields. In the case of small finite fields, it is more efficient to use the power form to represent field elements while applying the table lookup method for its arithmetic operations. Fast memory devices are good candidates for the implementation of tables as long as the table capacity does not exceed the current device technology. However, accomplishing addition operations in power form is an awkward task. Thus, Zech's logarithm with its special properties are presented to solve this nontrivial problem. This leads to the application of the index calculus method for finite field arithmetic. As the size of fields becomes larger, the conversion between power and polynomial forms creates a challenge task which is traditionally termed as the discrete logarithm and exponentiation problem. To avoid such a situation, polynomial form arithmetic becomes a commonly used method despite its difficulties in the multiplication operation. Three different algorithms presented in a step-by-step manner for multiplication are investigated based upon their respective basis representations (primal, dual, and normal polynomial form) of finite fields.

Finite digital convolution is a numerical procedure which has many powerful applications, such as the implementation of finite impulse response (FIR) and infinite impulse response (IIR) digital filters, the operation of auto- and cross-correlation, as well as the computations of the products of polynomials and very large integers. Chapter 4 detailed the general finite structure transforms necessary to carry out these applications. Recently, various researchers have proposed the use of transforms over finite structures using number theoretic concepts for error-free, fast, efficient computations of finite digital convolutions of real integer sequences or complex integer sequences. These number theorem transforms inherit a major disadvantage in terms of the requirement of a rigid relationship between the dynamic range and obtainable transform length. Fortunately, the transforms in finite structures also are applicable over extension fields which lead to the relaxation of the imposed limitations.

In many applications, such as radar or communication, pre-convolved data sequences consist of complex quantities. The transform allows greatly increased sample lengths over those defined in the ground field, GF(p), like those of FNT or MNT; for a sufficiently large p, one can use this transform to convert a sequence of complex integers a_n into the sequence A_k in GF(p^2) for which the inverse transform of A_k is precisely the original sequence of complex numbers a_n. Consequently, filtering operations or convolutions without round-off error are obtained using this transform on a sequence of complex integers. However, the application of the cyclic convolution property (CCP) of DFT serves to reduce the computational complexity of finite convolution only when some fast Fourier transform (FFT) algorithms are available and applied to the DFT.

For the higher order extension field, GF(p^m), transforms exist as long as the sufficient and necessary conditions exist. By applying finite field properties, such as the conjugacy property, and the basis representation of the field elements to the existing fast transform algorithms, a significant reduction in computational complexity is achieved. There also has been considerable work on high-speed residue number arithmetic for use in high data rate digital signal processing. The structure of a complex integer ring is generalized to form a complex residue number system (CRNS) which is a powerful computational number system. In CRNS, complex multiplication is accomplished by means of a real index calculus because the selection of the system parameters is not overly constrained by the algebraic structure required by a NTT. The CRNS is only useful when the complex variables are feasibly represented by the power form over a second order extension field and computed using the index calculus method. When complex variables are represented by a isomorphic mapping, traditionally referred to as the quadratic residue number system (QRNS), the usual operations are reduced to a point-wise operation. A further extension to the higher order of polynomial suggested by the algebraic integer concept is the polynomial RNS (PRNS) system. The principle advantage of the extended RNS processing is its ability to reduce a complex multiplication or addition to the calculation of a single integer multiplication or integer addition modulo a prime in parallel sets of residue channels. When the primes are small, the implementation complexity of the computation in each of the residue channels is correspondingly small — implying substantially high throughput.

All the finite field transforms and the RNS systems which are discussed perform over a ground field GF(p) where p may be a Fermat or Mersenne number, or p satisfies some specified condition such as having the form of $4k + 1$ or $2Nk + 1$. Without meeting these requirements, transforms may exist in the higher order extension field GF(p^m). A number of field properties, like index calculus, conjugacy in finite field elements, and basis representation, are used to perform and expedite these transforms in extension fields.

Chapter 5 elaborates the investigation of a fast finite field transform over GF(p^m) based on the Good-Thomas prime-factor FFT algorithm. The intermediate results are represented by using a normal basis representation. A significant computational reduction is obtained by applying a conjugacy relation to a cyclotomic coset of the intermediate variables, and by using the cyclic-shift property of p powers of the variables within the normal basis representation. Once the VLSI architecture for the butterfly module of a cyclotomic coset based on the algorithm is developed, these module arrays are used to form the stages of the fast transform. For the case of $p = 2$, $m = 8$, performance analysis shows a six-fold reduction in computational complexity which is achieved in terms of an added hardware budget.

A comparison between the total number of operations involved for the methods in their respective approach to the direct DFT which required approximately 65,000 operations clearly suggests a dramatic reduction in computational operations when the normal basis system is implemented. This system takes advantage of the multiplication-free shifting by applying the normal basis representation to the variables in the pipeline stages. In terms of physical realities, the simplified computational structure makes this system readily adaptable to a compact device without relinquishing any of its efficiency. Interfacing with pre-existing systems which have representations in a different format, such as the power form or the standard polynomial form, simply involves making a basis change at the end of the processing pipeline.

As fast Fourier transform (FFT) algorithms are applied over the finite field GF(p^m), the intermediate results are encoded in a normal basis representation. The evaluation of the intermediate results which are shown to be field elements of a cyclotomic coset within the finite field introduces a nontrivial multiply-accumulate task. To mitigate the problem, a fast and compact computational structure based on the basis-change algorithm is used to perform these operations over the finite field. This research effort suggests a new computational algorithm which involves the change of basis of field elements during the nontrivial evaluation of the coset elements. This finite field arithmetic is equipped with cyclic shifting for scaling, simple table lookups for polynomial reduction, and a series of component-wise modular adders. Application of these concepts to the existing fast transform not only serves to simplify the transform, but also leads to the development of a very compact signal processing device which can be implemented as a building block and seamlessly integrated to a discrete Fourier transform (DFT) system.

The application of the PRNS in complex multiplication offers tremendously low complexity within the digital signal processing (DSP) area. Unfortunately, isomorphic mappings between complex number and PRNS domains suffer from a nontrivial transform problem which eventually precludes the inherent advantages of the PRNS approach. Chapter 6 is a close examination of the issues involved in the development of the fast and compact third-order PRNS machine, especially the VLSI layout using the CMOS technology. A significant simplification in the mapping procedure is achieved by a FFT-like scheme and a sequence of primitive split-then-add operations. These operations originate from an algebraic congruence and a residue reduction of a Fermat prime within finite fields.

The complete development of the polynomial RNS system with its circuit and silicon layout not only suggests a significant reduction in both computation core and number system conversions but also confirms the advantages of the novel mapping algorithm. In terms of hardware budget, an N-order PRNS system equipped with the novel approach has a complexity on the order of $O(N)$. An algebraic integer processor of the same degree would be on the order of $O(N^2)$. Consequently, as N becomes larger to meet the requirement of high accuracy computation, a substantial saving is realized.

7.2 Future Research

Cozzens and Finkelstein [Coz85] introduced the idea of performing a computation in a certain ring of algebraic integers which is accomplished by approximating the appropriate complex roots of unity by elements of the ring. Games [Gam85] also illustrated a method to perform this approximation by elements of the algebraic integers of $\mathbf{Q}(\omega)$. Due to the nature of these approximations, the dynamic range of the computation is dramatically reduced which results in the advantageous application of the Polynomial RNS concept. When the primes of the PRNS are small, the implementation complexity of the computation in each of the residue channels is corresponding small which implies a substantially high throughput. Therefore, a worthy research task is to develop an efficient isomorphic mapping algorithm for higher order PRNS system to enable accurate complex arithmetic under the high degree algebraic integer approximation.

Massey and Omura [Wan85] suggested a multiplication scheme which performs the product of two elements in the finite field GF(p^2). While this system possesses the features of regularity and high-throughput, it is cumbersome (see Chapter 3). A finite field transform also was defined over GF(p^m) by Blahut [Bla83] which leads to this research work on the novel algorithm of applying the mathematical concepts of normal basis, conjugacy, and basis-change in finite fields. For real-time DSP applications, ultra high-speed hardware accelerators are required to implement the transform algorithms. In such a case, a bulky but high-throughput normal basis multiplier over GF(p^m) is a good candidate, but needs further development. Thus, another interesting area of research deals with solving the multiplicative F_{NB} function under the consideration of the defining polynomial of the finite field GF(p^m). Furthermore, since an array of F_{NB} function modules are required to implement an m-digit parallel normal basis multiplier, the future integrated circuit (IC) technology of wafer-scale integration may be the solution to this problem.

Finally, an interesting historical problem which relates to the finite field transform of DFT structure and CCP property is to increase the dynamic range requirement for digital convolution. Jenkins [Jen80] proposed a direct sum of L second degree extension fields for wide range complex arithmetic which is similar to the application of the CRT to complex integer convolution by Reed and Trung [Ree75b]. They show the existence of the isomorphism of

$$\mathrm{GF}(p^2) \simeq \mathrm{GF}(p_1^2) + \mathrm{GF}(p_2^2) + \ldots + \mathrm{GF}(p_L^2) \tag{7.1}$$

where $p = p_1 \cdot p_2 \cdots p_L$, p_i are primes such that -1 is a quadratic nonresidue. For the higher degree extension of $\mathrm{GF}(p^m)$ where $p = p_1 \cdot p_2 \cdots p_L$, p_i are primes, it is required to find the following similar relationships:

$$\mathrm{GF}(p^m) \simeq \mathrm{GF}(p_1^m) + \mathrm{GF}(p_2^m) + \ldots + \mathrm{GF}(p_L^m); \tag{7.2}$$

Suppose $\alpha_1, \alpha_2, \ldots, \alpha_L$ are primitive dth roots of unity in $\mathrm{GF}(p_1^m)$, $\mathrm{GF}(p_2^m)$, . . . ,$\mathrm{GF}(p_L^m)$, respectively. Hence, $(\alpha_1^d, \alpha_2^d, \ldots, \alpha_L^d) = (1, 1, \ldots, 1)$ is the unity element in the external direct product of the finite field $\mathrm{GF}(p^m)$ as shown in Equation (7.2) where $d \mid p_i^m$ for all $i = 1, 2, \ldots, L$ and $(\alpha_1, \alpha_2, \ldots, \alpha_L)$ corresponds to an element $\alpha \in \mathrm{GF}(p^m)$ such that α is a dth root of unity in $\mathrm{GF}(p^m)$. Hence, applying the CRT to the finite field transform pairs as shown in Equations (4.44) and (4.45), the relationship holds

$$\begin{aligned}
&(A_{1k}, A_{2k}, \ldots, A_{Lk}) \\
&= \sum_{n=0}^{d-1} (a_{1n}, a_{2n}, \ldots, a_{Ln})\,(\alpha_1, \alpha_2, \ldots, \alpha_L)^{nk} \\
&= \Big(\sum_{n=0}^{d-1} a_{1n}\alpha_1^{nk}, \sum_{n=0}^{d-1} a_{2n}\alpha_2^{nk}, \ldots, \sum_{n=0}^{d-1} a_{Ln}\alpha_L^{nk}\Big).
\end{aligned} \tag{7.3}$$

Therefore, the technique which exhibits the highest degree of parallelism improves the dynamic range problem effectively and the regular architecture of each channel is readily suited for the VLSI implementation.

APPENDIX A
ALGEBRAIC FOUNDATIONS: DEFINITIONS AND THEOREMS

A.1 GROUPS

D1.1. A group is a set G together with a binary operation $*$ on G, that assigns to every two elements $g, h \in G$ an element $g * h$ of G, such that the following axioms hold:

a. The associative law

b. Existence of an identity element

c. Existence of the inverse

If the group also satisfies $g * h = h * g$, then the group is called abelian (or commutative).

D1.2. A multiplicative group G is said to be cyclic if there is an element $\alpha \in G$ such that for any $b \in G$ there is some integer i with $b = \alpha^i$. Such an element α is called a generator of the cyclic group, and we write $G = <\alpha>$.

D1.3. For arbitrary integer a, b and a positive integer n, we say that a is congruent to b modulo m, and write $a \equiv b \bmod n$, if the difference $a - b$ is a multiple of n, that is, if $a = b + kn$ for some integer k. Consider the equivalence classes into which the relation of the congruence modulo n partitions the set **Z**. These will be the sets

$$0 + n\mathbf{Z} = \{ \ldots, -2n, -n, 0, n, 2n, \ldots \}$$

$$1 + n\mathbf{Z} = \{ \ldots, 1-2n, 1-n, 1, 1+n, 1+2n, \ldots \}$$

$$\vdots$$

$$(n-1) + n\mathbf{Z} = \{ \ldots, -n-1, -1, n-1, 2n-1, 3n-1, \ldots \}$$

D1.4. The group formed by the set $\{ n\mathbf{Z}, 1 + n\mathbf{Z}, \ldots, (n-1) + n\mathbf{Z} \}$ of equivalence classes modulo n with the operation $(a + n\mathbf{Z}) + (b + n\mathbf{Z}) = (a + b + n\mathbf{Z})$ is called the group of integers modulo n, and denoted by $\mathbf{Z}_n$.

D1.5. A group G is called finite if it contains finitely many elements. The number of elements in a finite group is called its order and denoted $|G|$.

D1.6. A mapping $f : G \rightarrow H$ of the group G into H is called a homomorphism of G into H if f preserves the operation of G. That is, if $*$ and $\cdot$ are the operations of G and H respectively, then for all $a, b \in G$ we have $f(a * b) = f(a) \cdot f(b)$. In addition, if f is onto H, then f is called an homomorphism onto. If f is a one-to-one homomorphism of G onto H, then f is called an isomorphism and we say that G and H are isomorphic. An isomorphism of G onto G is called an automorphism.

D1.7. The kernel of the homomorphism $f : G \rightarrow H$ of the group G into the group H is the set $\ker f = \{ a \in G : f(a) = e' \}$ where e' is the identity element in H.

D1.8. A subgroup H of a group G is called a normal subgroup if $gH = Hg$ for all $g \in G$.

D1.9. The left cosets of G modulo H are denoted by $aH = \{ ah : h \in H \}$, where a is a fixed element of G.

D1.10. For a normal subgroup H of G, the group formed by the left cosets of G modulo H is called the quotient group or factor group of G modulo H and denoted by G/H.

A.2 RINGS AND FIELDS

D2.1. A ring $(R, +, \cdot)$ is a set R, together with two binary operations, denoted by $+$ and $\cdot$, such that

a. $(R, +)$ is an abelian group

b. The multiplication ‘ $\cdot$ ’ is associative

c. The distributive laws hold, that is for all $a, b, c \in R$

$$a \cdot (b+c) = a \cdot b + a \cdot c,$$

$$(b+c) \cdot a = b \cdot a + c \cdot a.$$

D2.2. A ring is called a ring with identity if the ring has a multiplicative identity. If the multiplication is commutative then the ring is called a commutative ring. A ring is called an integral domain if it is a commutative ring with identity $e \neq 0$ in which $ab = 0$ implies $a = 0$ or $b = 0$.

D2.3. A subset I of a ring R is called an ideal provided I is a subring of R for all $a \in I$ and $r \in R$ we have $ar \in I$ and $ra \in I$.

D2.4. Let R be a commutative ring. An ideal I of R is said to be principal if there is an $a \in R$ such that $I = (a)$, where $(a) = \{ ra : r \in R \}$. In this case, I is also called the principal ideal generated by a.

D2.5. The ring of residue classes of the ring R modulo the ideal I under the operations

$$(\mathrm{a} + I) + (\mathrm{b} + I) = (\mathrm{a} + \mathrm{b}) + I,$$

$$(\mathrm{a} + I) \cdot (\mathrm{b} + I) = (\mathrm{a} \cdot \mathrm{b}) + I$$

is called the residue class ring of R modulo I and is denoted by R/I.

D2.6. For a prime p, let F_p be the set $\{ 0, 1, \ldots, p-1 \}$ of integers and let $\phi : \mathbf{Z}/(p) \rightarrow F_p$ be the mapping defined by $\phi (a + n\mathbf{Z}) = a$ for $a = 0, 1, \ldots, p-1$. Then F_p, endowed with the field structure induced by ϕ, is a finite field called the Galois field of order p.

D2.7. For a finite field F_p we denote by F_p^* the multiplicative group of nonzero elements of F_p.

D2.8. A generator of the cyclic group F_p^* is called a primitive element of F_p.

D2.9. A commutative ring R with identity is called a field if and only if $R\backslash\{0\}$ is a group under multiplication, where $R\backslash\{0\} = \{ a : a \in R, a \neq 0 \}$.

D2.10. If R is an arbitrary ring and there exists a positive integer n such that $nr = 0$ for every $r \in R$, then the least such positive integer n is called the characteristic of R. If no such integer n exists, R is said to have characteristic 0.

D2.11. The ring formed by the polynomials over R is called the polynomial ring over R and denoted by $R[x]$.

D2.12. Let

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

be a polynomial over R which is not the zero polynomial, so that we can suppose $a_n \neq 0$. Then a_n is called the leading coefficient of $f(x)$ and a_0 the constant term, which n is called the degree of $f(x)$, in symbol $n = deg\ (f(x)) = deg(f)$. If the leading coefficient of $f(x)$ is 1, then $f(x)$ is called a monic polynomial.

D2.13. A polynomial $p \in F[x]$ is said to be irreducible over F if p has positive degree and $p = bc$ with $b, c \in F[x]$ implies that either b or c is a constant polynomial.

D2.14. An element $b \in F$ is called a root or zero of the polynomial $f \in F[x]$ if $f(b) = 0$.

D2.15. A field containing no proper subfield is called a prime field.

D2.16. Let K be a subfield of F and M any subset of F. Then the field $K(M)$ is defined as the intersection of all subfields of F containing both K and M and is called the extension field of K obtained by adjoining the elements in M. For finite $M = \{ \theta_1, \ldots, \theta_n \}$ we write $K(M) = K(\theta_1, \ldots, \theta_n)$. If M consists of a single element $\theta \in F$, then $K(\theta)$ is said to be a single extension of K and θ is called a defining element of $K(\theta)$ over K.

D2.17. Let K be a subfield of F and $\theta \in F$. If θ satisfies a nontrivial polynomial equation over K, that is, if $a_n\theta^n + \ldots + a_1\theta + a_0 = 0$ with $a_i \in K$ not all being 0, then θ is said to be *algebraic* over K. An extension $K(\theta)$ of K is called algebraic over K if every element of $K(\theta)$ is algebraic over K.

D2.18. If $\theta \in F$ is algebraic over K, then the uniquely determined monic polynomial $g \in K[x]$ generating the ideal $I = \{ f \in K[x] : f(\theta) = 0 \}$ is called the minimal polynomial (or defining polynomial) of θ over K. By the degree of θ over K we mean the degree of g.

D2.19. If $K(\theta\ )$ is finite dimensional, then $K(\theta\ )$ is called a finite extension of K. The dimension of the vector space $K(\theta\ )$ is called the degree of $K(\theta\ )$ over K, in symbol $[K(\theta\ ) : K]$

D2.20. Let $\theta \in K[x]$ be of positive degree and F an extension field of K. Then f is said to split in F if f can be written as a product of linear factor in $F[x]$ – that is, if there exist elements α_1, $\alpha_2, \ldots, \alpha_n \in F$ such that $f(x) = a(x-\alpha_1)(x-\alpha_2)\ldots(x-\alpha_n)$, where a is the leading coefficient of f. The field F is a splitting field of f over K if f splits in F .

D2.21. Let F_{p^n} be an extension of F_p and let $\alpha \in F_{p^n}$. Then the elements α , α^p, α^{p^2}, . . ., $\alpha^{p^{n-1}}$ are called the conjugates of α with respect to F_p.

D2.22. Let m be a positive integer. The splitting field of $x^m - 1$ over a field K is called the mth cyclotomic field over K and denoted by $K^{(m)}$. The roots of $x^m - 1$ in $K^{(m)}$ are called the mth roots of unity over K and the set of all these roots is denoted by $E^{(m)}$.

D2.23. Let K be a field of characteristic p, m a positive integer not divisible by p, and α a primitive mth root of unity over K. Then the polynomial

$$Q_m(x) = \sum_{s=1}^{m}(x - \alpha^s)$$

where $\gcd(s, m) = 1$, is called the mth cyclotomic polynomial over K.

D2.24. (Roots of unity and cyclotomic polynomials) According to D2.22, a special case is obtained if K is the field of rational numbers. Then $K^{(m)}$ is a subfield of the field of complex numbers. For our purposes, the most important case is that of a finite field K. The polynomial $Q_m(x)$ is clearly independent of the choice of α . The degree of $Q_m(x)$ is $\phi\ (m)$ and its coefficients obviously belong to the mth cyclotomic field over K.

A.3 THEOREMS

TH3.1. If F is a finite field with q elements and K is a subfield of F, then the polynomial $x^q - x$ in $K[x]$ as

$$x^q - x = \sum_{a \in F} (x - a)$$

and F is a splitting field of $x^q - x$ over K.

<u>TH3.2.</u> (Existence and uniqueness of finite fields) For every prime p and every positive integer n there exists a finite field with p^n elements. Any finite field with $q = p^n$ elements is isomorphic to the splitting field of $x^q - x$ over F_p. We should denote this field by F_p or GF(p).

<u>TH3.3.</u> If f is an irreducible polynomial in $F_p[x]$ of degree n, then f has a root α in F_{p^n}. Furthermore, all the roots of f are simple and are given by the n distinct elements α , α^p, α^{p^2}, . . . , $\alpha^{p^{n-1}}$ of F_{p^n}.

APPENDIX B
CIRCUIT DESIGNS: HP-DCS AND MAGIC CAD TOOLS

The documentation reported in this Appendix include detailed design schematic, Postscript printout of the Magic layouts, and HP-DVI timing diagrams of the third-order polynomial RNS processor. At the last part of this Appendix, examples of the Caltech Intermediate Form (CIF) file and the Postscript file of a two-input NAND gate are demonstrated.

B.1 The HP-DCS Schematics of the Nine-Bit Third-Order PRNS Processor

1) mux9 – The Nine-Bit Multiplexer;

2) lat4b – The Four-Tuple Latch;

3) mul8x8 – The Eight-Bit by Eight-Bit Multiplier;

4) scalerNRSQ – The $(-r^2)$-Scaler Module; and

5) scalerNR – The $(-r)$-Scaler Module.

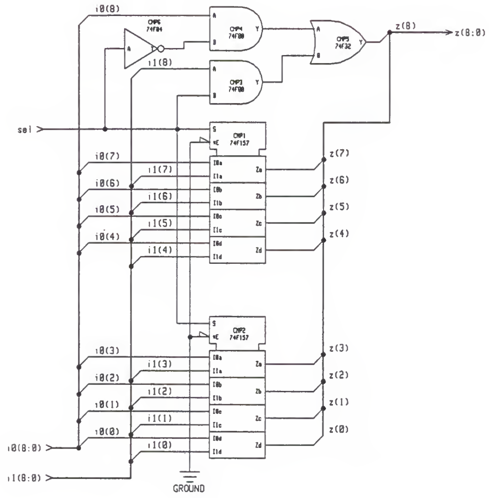


Figure B.1.1 mux9 – The Nine-Bit Multiplexer

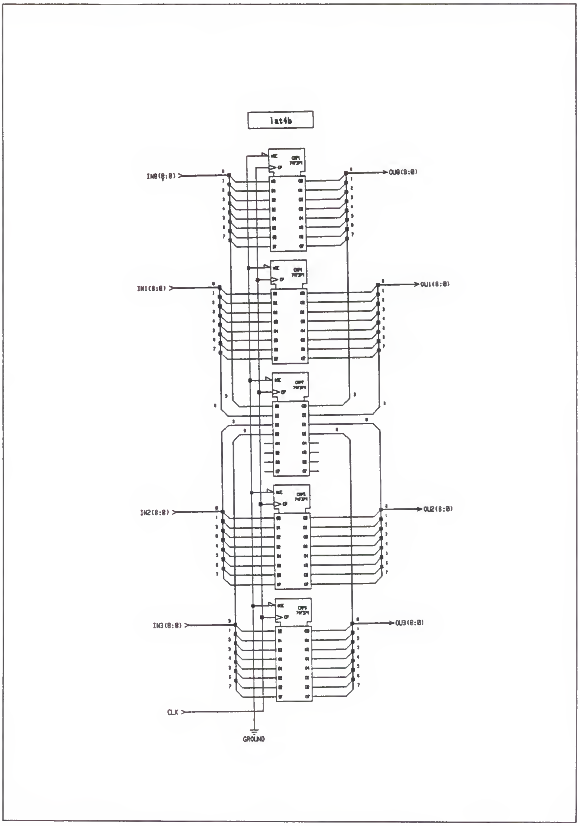


Figure B.1.2 lat4b – The Four-Tuple Latch

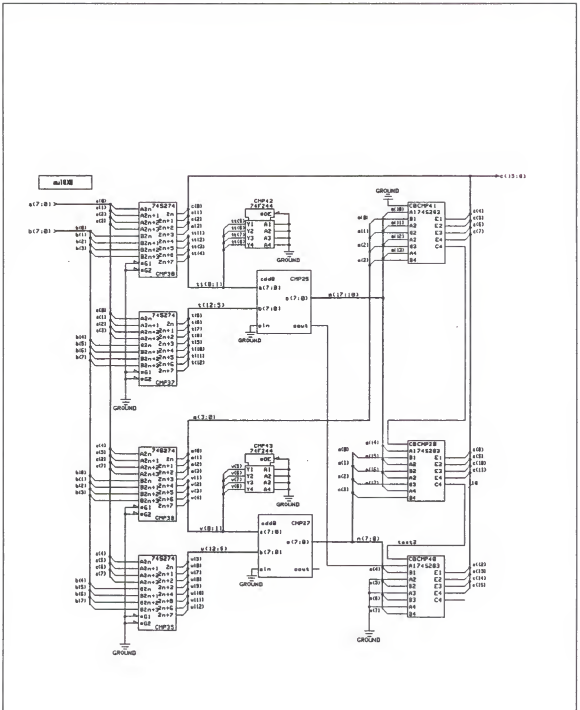


Figure B.1.3 mul8x8 – The Eight-Bit by Eight-Bit Multiplier

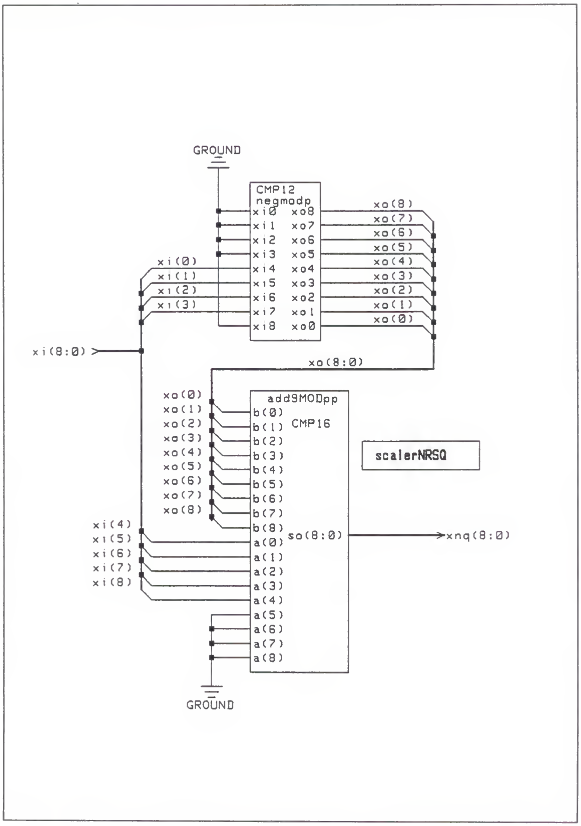

GROUND
CMP12
negmodp
xi0 xi1 xi2 xi3 xi4 xi5 xi6 xi7 xi8
xo8 xo7 xo6 xo5 xo4 xo3 xo2 xo1 xo0
xo(8) xo(7) xo(6) xo(5) xo(4) xo(3) xo(2) xo(1) xo(0)
xi(0) xi(1) xi(2) xi(3)
xi(8:0)
xo(8:0)
add9MODpp
CMP16
b(0) b(1) b(2) b(3) b(4) b(5) b(6) b(7) b(8)
a(0) a(1) a(2) a(3) a(4) a(5) a(6) a(7) a(8)
xi(4) xi(5) xi(6) xi(7) xi(8)
so(8:0)
xnq(8:0)
scalerNRSQ
GROUND

Figure B.1.4 scalerNRSQ – The ($-r^2$)-Scaler Module

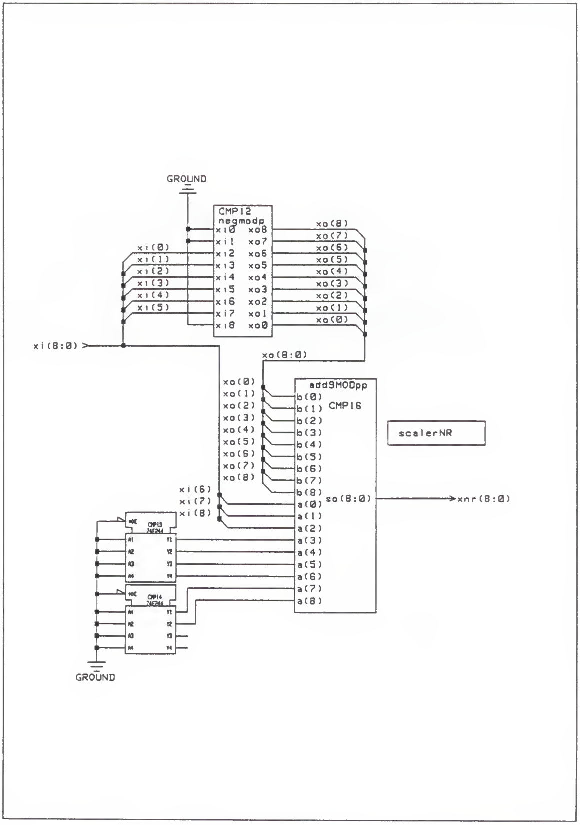

GROUND
CMP12
negmodp
xi0 xi1 xi2 xi3 xi4 xi5 xi6 xi7 xi8
xo8 xo7 xo6 xo5 xo4 xo3 xo2 xo1 xo0
xi(0) xi(1) xi(2) xi(3) xi(4) xi(5)
xo(8) xo(7) xo(6) xo(5) xo(4) xo(3) xo(2) xo(1) xo(0)
xi(8:0)
xo(8:0)
add9MODpp
CMP16
b(0) b(1) b(2) b(3) b(4) b(5) b(6) b(7) b(8)
a(0) a(1) a(2) a(3) a(4) a(5) a(6) a(7) a(8)
so(8:0)
xnr(8:0)
scalerNR
xi(6) xi(7) xi(8)
CMP13
74F244
CMP14
74F244
GROUND

Figure B.1.5 scalerNR – The (–*r*)-Scaler Module

B.2 The HP-DVI Timing Diagrams of the Nine-Bit Third-Order PRNS Processor

1) The Timing Diagram of negmodp – The Negator;

2) The Timing Diagram of summodp – The Sum Modulo Converter;

3) The Timing Diagram of sumovck – The Sum Overflow Checker;

4) The Timing Diagram of mux9 – The Nine-bit Multiplexer;

5) The Timing Diagram of add9modp – The Nine-bit Modulo Adder;

6) The Timing Diagram of mul8x8 – The Eight-Bit by Eight-Bit Multiplier;

7) The Timing Diagram of mul9modp – The Nine-Bit Modulo Multiplier;

8) The Timing Diagram of forwmap – The Isomorphic Forward Mapping Module;

9) The Timing Diagram of backmap – The Isomorphic Inverse Mapping Module; and

10) The Timing Diagram of scalerNRSQ – The $(-r^2)$-Scaler Module.

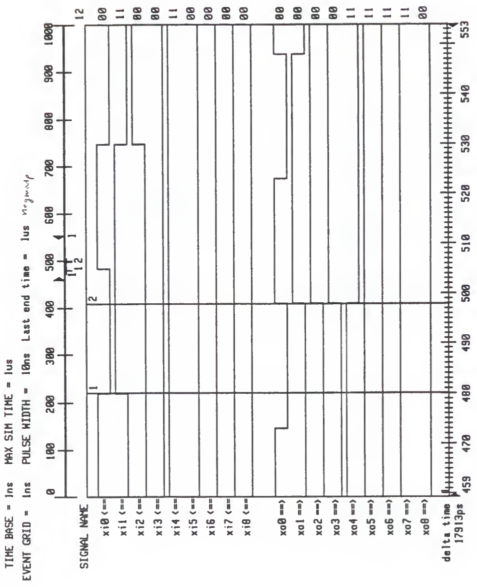


Figure B.2.1 The Timing Diagram of negmodp – The Negator

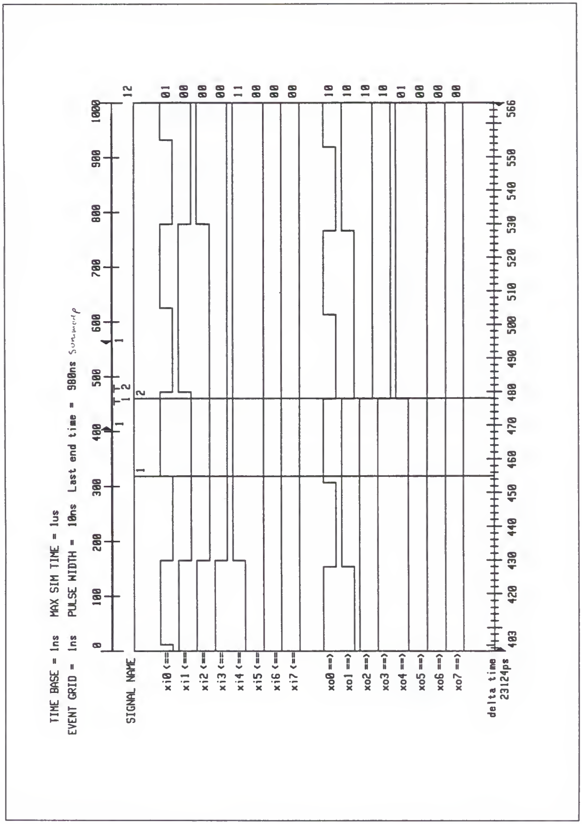


Figure B.2.2 The Timing Diagram of summodp – The Sum Modulo Converter

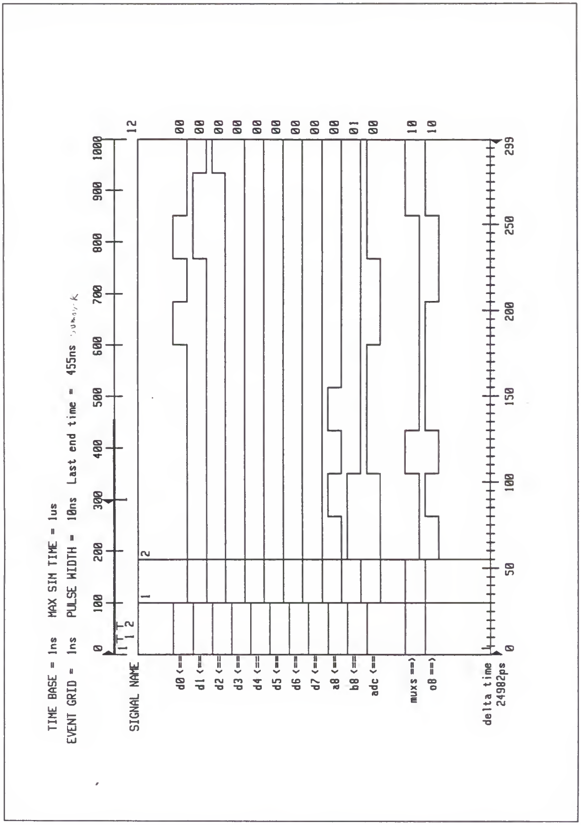


Figure B.2.3 The Timing Diagram of sumovck – The Sum Overflow Checker

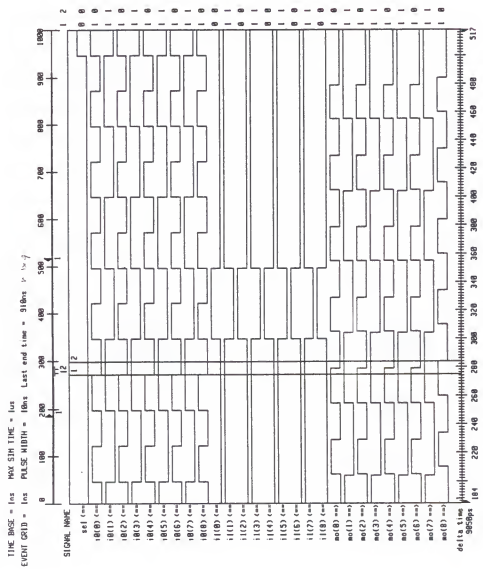


Figure B.2.4 The Timing Diagram of mux9 – The nine-bit multiplexer

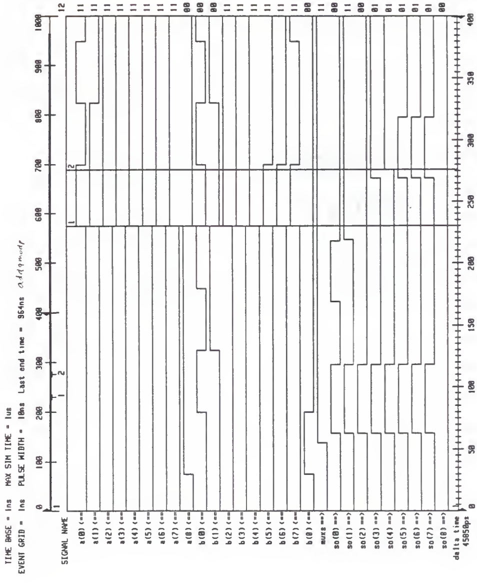


Figure B.2.5 The Timing Diagram of add9modp – The Nine-bit Modulo Adder

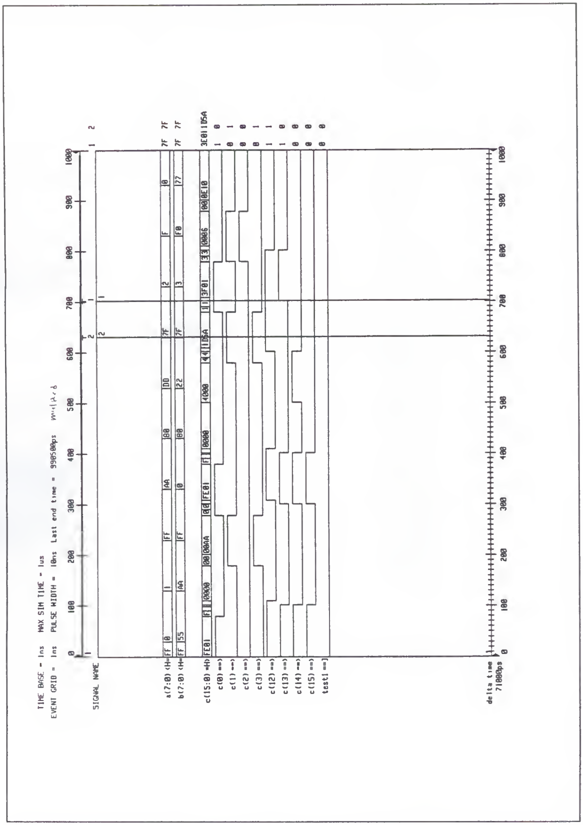


Figure B.2.6 The Timing Diagram of mul8x8 – The Eight-Bit by Eight-Bit Multiplier

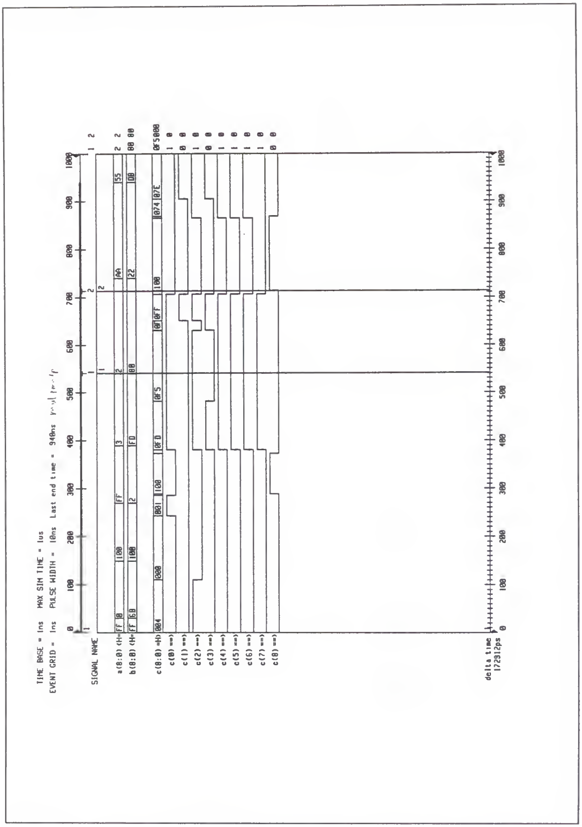


Figure B.2.7 The Timing Diagram of mul9modp – The Nine-Bit Modulo Multiplier

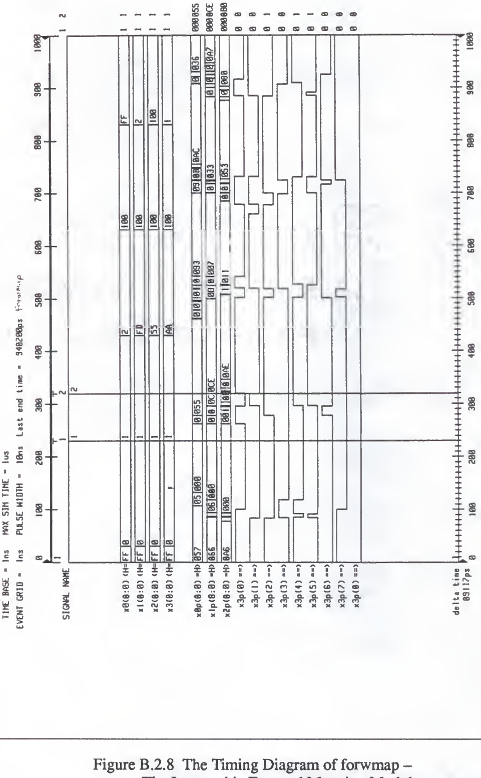


Figure B.2.8 The Timing Diagram of forwmap –
The Isomorphic Forward Mapping Module

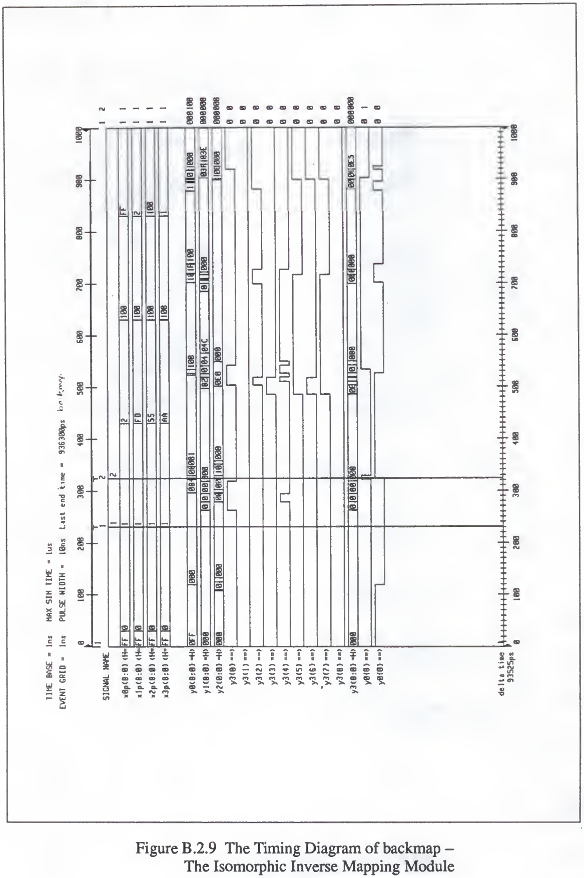


Figure B.2.9 The Timing Diagram of backmap –
The Isomorphic Inverse Mapping Module

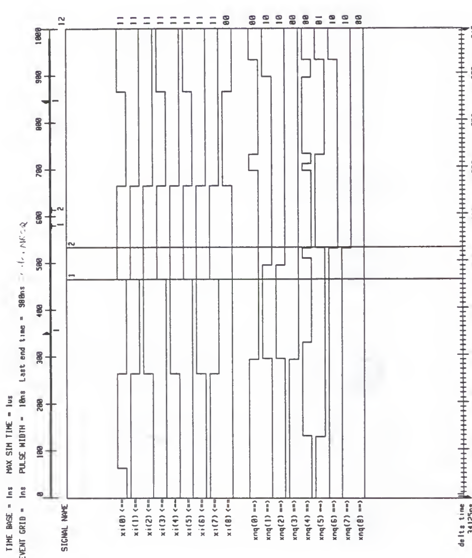


Figure B.2.10 The Timing Diagram of scalerNRSQ – The $(-r^2)$-scaler module

B.3 The Magic Design of the Five-Bit Third-Order PRNS Processor

1) The Magic Design Circuit diagram I;

2) The Magic Design Circuit diagram II;

3) The Magic Design Circuit diagram III;

4) The Magic Design Circuit diagram IV;

5) The Magic Design Circuit diagram V;

6) The Magic Transistor Layout of multip – The Three by Three Multiplier;

7) The Magic Transistor Layout of sumall – The Modular Adder;

8) The Magic Transistor Layout of smod – The Modular Adder for the isomorphic mappings;

9) The Magic Transistor Layout of scln – The FFT-Type module for the isomorphic mappings;

10) The Magic Transistor Layout of fmap – The Isomorphic forward mappings;

11) The Magic Transistor Layout of padin – The Input Pad; and

12) The Magic Transistor Layout of padout – The Output Pad.

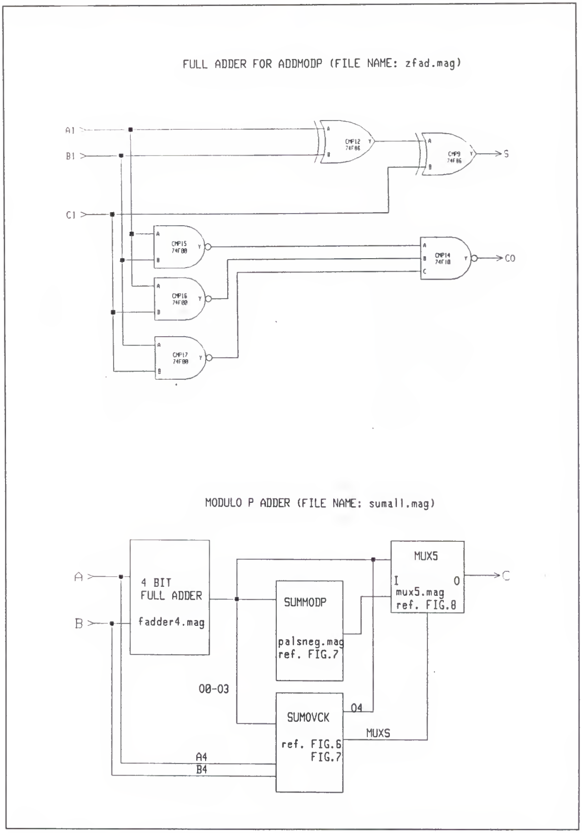


Figure B.3.1 The Magic Design Circuit diagram I

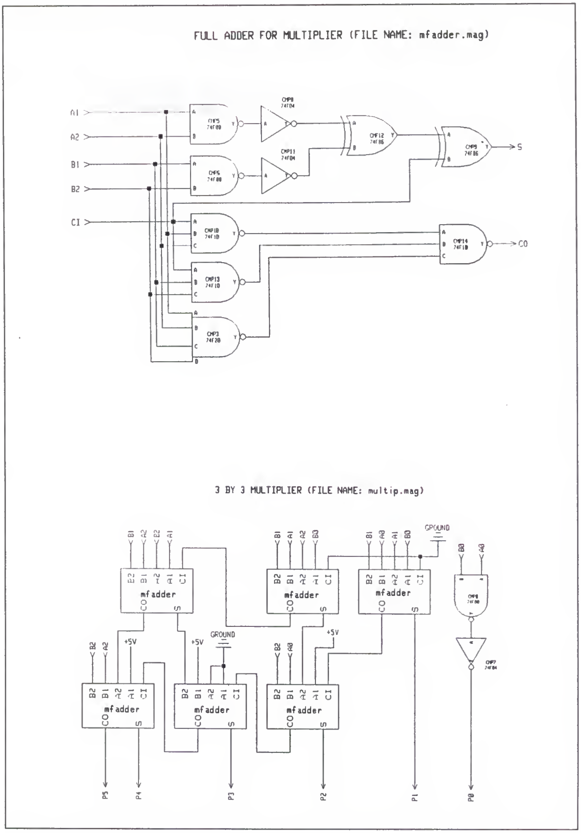

FULL ADDER FOR MULTIPLIER (FILE NAME: mfadder.mag)
A1
A2
B1
B2
CI
S
CO
74F00
74F04
74F86
74F10
74F20
3 BY 3 MULTIPLIER (FILE NAME: multip.mag)
mfadder
GROUND
+5V
P5
P4
P3
P2
P1
P0

Figure B.3.2 The Magic Design Circuit diagram II

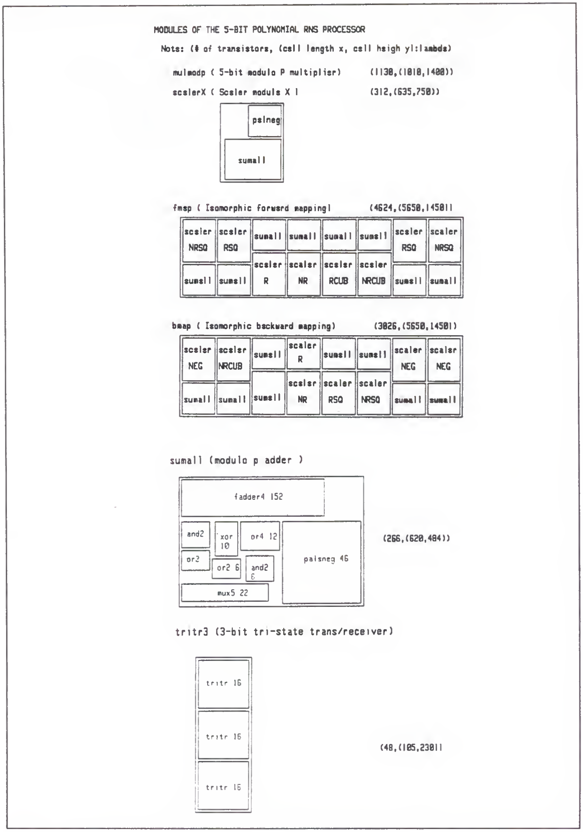

MODULES OF THE 5-BIT POLYNOMIAL RNS PROCESSOR
Note: (# of transistors, (cell length x, cell heigh y):lambda)
mulmodp (5-bit modulo P multiplier) (1130,(1010,1400))
scalerX (Scaler module X) (312,(635,750))
palneg
sumall
fmap (Isomorphic forward mapping) (4624,(5650,1450))
scaler NRSQ
scaler RSQ
sumall
scaler R
scaler NR
scaler RCUB
scaler NRCUB
bmap (Isomorphic backward mapping) (3026,(5650,1450))
scaler NEG
scaler NRCUB
scaler R
scaler NR
scaler RSQ
scaler NRSQ
sumall (modulo p adder)
fadder4 152
and2
xor 10
or4 12
or2
or2 6
and2 6
palsneg 46
mux5 22
(266,(620,484))
tritr3 (3-bit tri-state trans/receiver)
tritr 16
tritr 16
tritr 16
(48,(105,230))

Figure B.3.3 The Magic Design Circuit diagram III

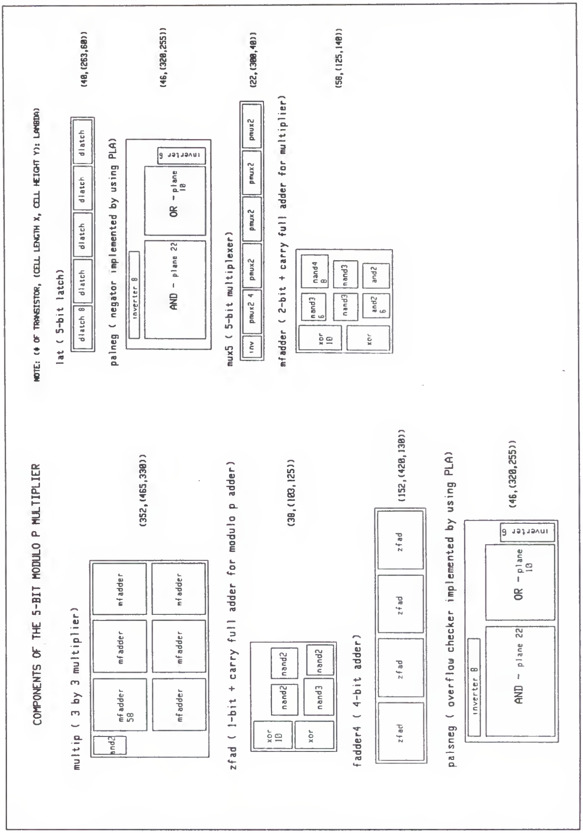

COMPONENTS OF THE 5-BIT MODULO P MULTIPLIER
NOTE: (# OF TRANSISTOR, (CELL LENGTH X, CELL HEIGHT Y): LAMBDA)
multip (3 by 3 multiplier)
(352,(465,330))
and2
mfadder 58
mfadder
zfad (1-bit + carry full adder for modulo p adder)
(38,(183,125))
xor 10
xor
nand2
nand3
fadder4 (4-bit adder)
(152,(420,130))
zfad
palsneg (overflow checker implemented by using PLA)
(46,(320,255))
inverter 8
AND - plane 22
OR - plane 10
inverter 6
lat (5-bit latch)
(40,(263,60))
dlatch 8
dlatch
palneg (negator implemented by using PLA)
(46,(320,255))
mux5 (5-bit multiplexer)
(22,(300,40))
inv
pmux2 4
pmux2
mfadder (2-bit + carry full adder for multiplier)
(58,(125,140))
nand3 6
nand4 8
and2 6
and2

Figure B.3.4 The Magic Design Circuit diagram IV

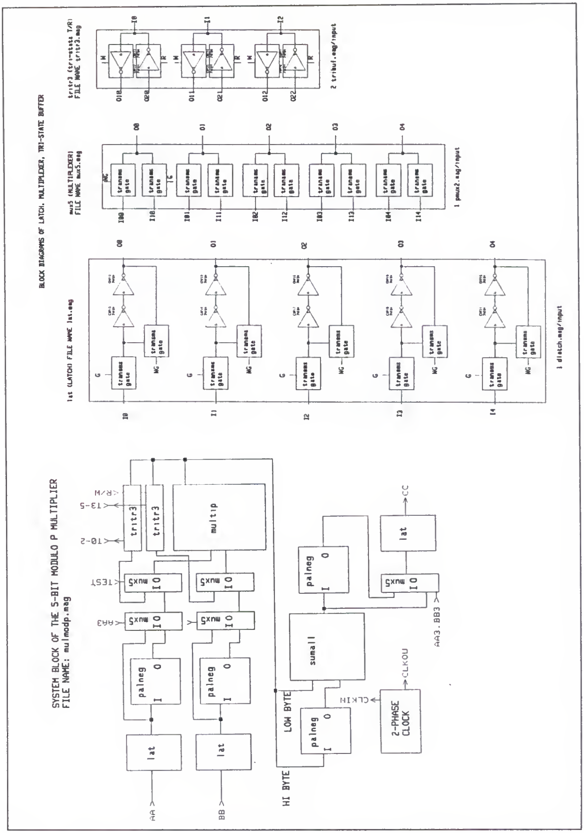

SYSTEM BLOCK OF THE 5-BIT MODULO P MULTIPLIER
FILE NAME: mulmodp.mag
BLOCK DIAGRAMS OF LATCH, MULTIPLEXER, TRI-STATE BUFFER
lat (LATCH) FILE NAME lat.mag
mux5 (MULTIPLEXER) FILE NAME mux5.mag
tritr3 (tri-state T/R) FILE NAME tritr3.mag
1 dlatch.mag/input
1 pmux2.mag/input
2 tribuf.mag/input
trans gate
lat
palneg
mux5
tritr3
multip
sumall
2-PHASE CLOCK
LOW BYTE
HI BYTE
AA
BB
CC
TEST
AA3
AA3.BB3
CLKIN
CLKOU
T0-2
T3-5
R/W

Figure B.3.5 The Magic Design Circuit diagram V

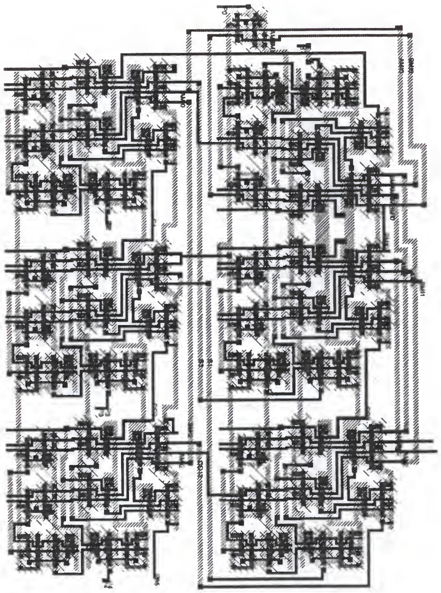


Figure B.3.6 The Magic Transistor Layout of multip – The Three by Three Multiplier

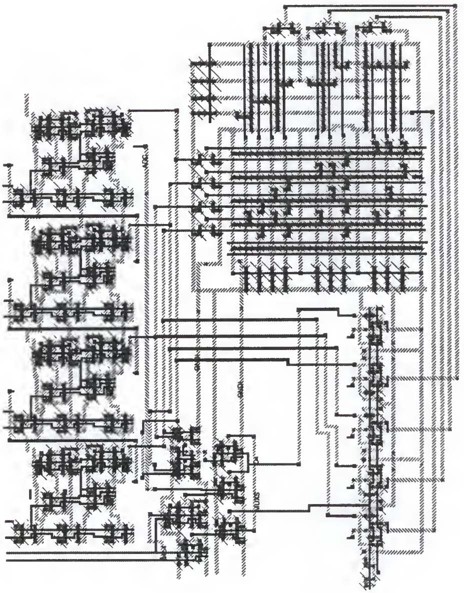


Figure B.3.7 The Magic Transistor Layout of sumall – The Modular Adder

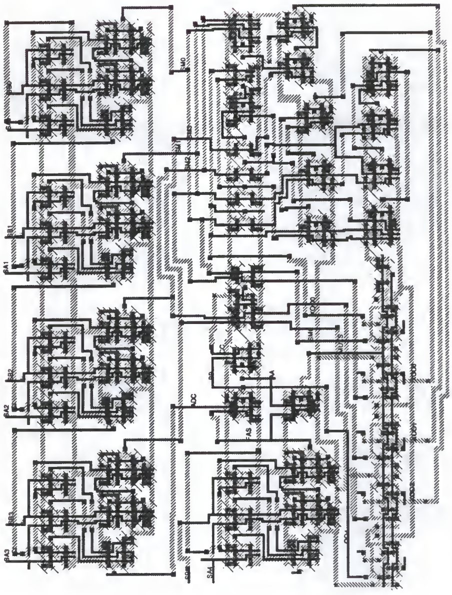


Figure B.3.8 The Magic Transistor Layout of smod –
The Modular Adder for the isomorphic mappings

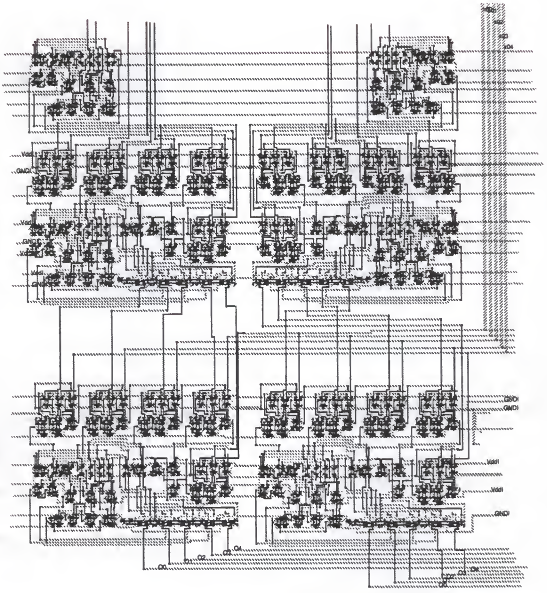


Figure B.3.9 The Magic Transistor Layout of scln –
The FFT-Type module for the isomorphic mappings

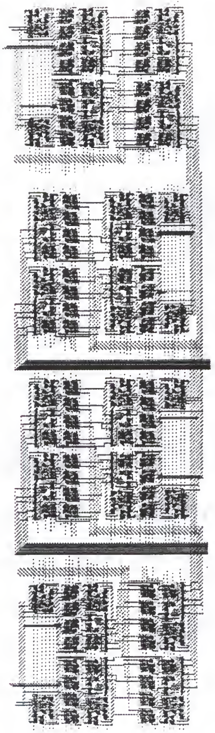


Figure B.3.10 The Magic Transistor Layout of fmap –
The Isomorphic forward mappings

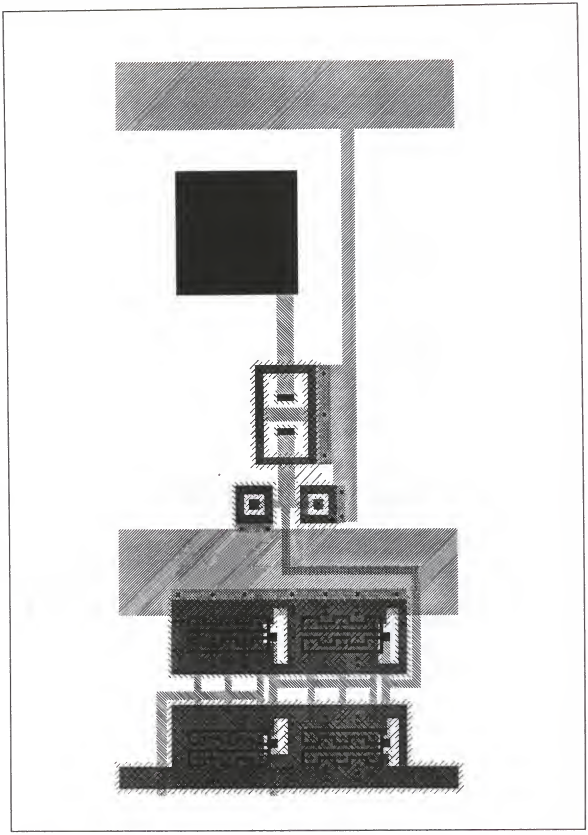


Figure B.3.11 The Magic Transistor Layout of padin – The Input Pad

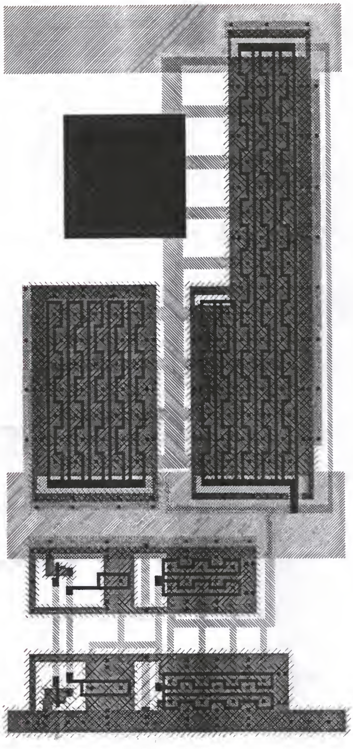


Figure B.3.12 The Magic Transistor Layout of padout – The Output Pad

B.4 CIF and Postscript Files of a Two-Input NAND Gate

B.4.1 CIF: A Geometry Language

CIF, the Caltech Intermediate Form is a language whose primitives are colored shapes. It is called "intermediate" because it is both a language in which designs of circuits can be couched and a language from which the masks (like negatives) from which a chip is fabricated can be made automatically [Mea80a]. A brief discussion of the language and an example of a two-input NAND gate are presented.

(1) The Box Statement

The most basic statement is one defining a rectangle, called a box. We place a box at a particular place on a grid by a statement of the form

B *<xdist>* *<ydist>* *<xcent>* *<ycent>*

where *<xdist>* and *<ydist>* are the lengths of the sides along the x and y axes; and *<xcent>* *<ycent>* are the x and y coordinates of the center of the rectangle. We can also specify a rotation of the rectangle with two additional components. The clause

D a b

calls for the x axis to be rotated until it has slope a/b. Thus, a box of length 6, width 2, center at (1, 3) and being rotated forty-five degrees clockwise is specified as

B 6 2 1 3 D –1 1.

(2) The Layer Statement

A box must be assigned a "color" or "layer" in addition to the box statement. The statement

L *<layer designation>*

specifies a layer. The *<layer designation>* is a code for the name of one of the layers used in the design. The most commonly used ones in the CMOS technology are CWN (nwell), CWP (pwell), CMS (allMetal2), CAA (allDiff), CCA (ndc,pdc) and CCP (pc).

(3) Defined Cells

CIF has a mechanism much like a procedure call, that enables us to define a cell, or symbol, to be some collection of the shapes, and then to "call" that cell as many times as we wish. The definition of a cell is introduced by a statement of the form

DS *<symbol number> <scale>*

where the *<symbol number>* is an integer that serves as the "name" of the cell, and *<scale>* is a pair of integers *a* and *b*, such that all dimensions and coordinates of boxes are multiplied by *a/b*. The DS stands for "Definition Start" whereas the end of a definition is marked by the statement DF, or "definition Finish." The fundamental unit of dimensions is not λ but 0.01 micron. Thus, if λ = 1.5 microns then *<scale>* = 150 1.

The call of a symbol is effected by the statement

C *<symbol number> <list of transformations>*

where *<list of transformations>* is a list of elements, each with one of the forms (a) T *<xorigin> <yorigin>*. (b) R *a b*. (c) MX and/or MY. These forms stand for Translate, Rotate, and Mirror respectively. An CIF example of a two-input NAND gate is demonstrated in FigureB.4.1.

```
DS 1 50 2;
9 nand2;
L CWN;
   B 104 8 92 40;
   B 120 68 92 2;
L CMF;
   B 80 16 92 24;
   B 16 28 60 2;
   B 16 16 92 -4;
   B 16 28 124 2;
   B 12 12 94 -18;
   B 44 12 110 -30;
   B 12 16 126 -44;
   B 32 28 68 -66;
   B 16 16 124 -60;
   B 80 16 92 -88;
L CPG;
   B 8 72 76 -4;
   B 24 8 84 -44;
   B 8 48 92 -72;
   B 8 128 108 -32;
L CAA;
   B 16 28 60 18;
   B 16 28 124 18;
   B 80 16 92 -4;
   B 80 16 92 -60;
L CCA;
   B 8 8 60 -4;
   B 8 8 92 -4;
   B 8 8 124 -4;
   B 8 8 76 -60;
   B 8 8 124 -60;
L CCA;
   B 8 8 60 24;
   B 8 8 124 24;
   B 8 8 60 -60;
L CSN;
   B 32 24 60 28;
   B 32 24 124 28;
   B 72 32 104 -60;
L CSP;
   B 32 4 60 14;
   B 32 4 124 14;
   B 96 32 92 -4;
   B 24 32 56 -60;
94 O 124 -32 CMF;
94 A 76 -44 CPG;
94 B 108 -44 CPG;
94 GND! 76 -76 CMF;
94 Vdd! 92 24 CMF;
DF;
C 1;
End
```

Figure B.4.1 The CIF Description of A Two-Input NAND Gate

B.4.2 PostScript: A Geometry Language

```
%!PS-Adobe-1.0
%%Creator: cif2ps
%%DocumentFonts: Helvetica
%%Pages: (atend)
%%EndComments
% PostScript from "cif2ps," a CIF to PostScript translator by:
%           Arthur Simoneau, The Aerospace Corporation, El Segundo, Calif
% with additions by:
%           Marc Lesure, Arizona State University, Tempe, AZ (May 1988)
% as well as:
%           Gordon W. Ross, The MITRE Corporation, Bedford, MA (June 1989)
%                       (proud author of this header code :-)

% Header code follows:

/B {        % dx dy x1 y1 ==> ---
            % Build a box: size=(dx,dy); lower_left=(x1,y1)
            newpath moveto dup 0 exch         % dx dy 0 dy
            rlineto exch 0                                  % dy dx 0
            rlineto 0 exch neg                % 0 -dy
            rlineto closepath
} bind def % B

/L {        % step angle ==> ---
            % Fill the current path with lines spaced by STEP and
            % rotated from the X-axis by ANGLE degrees ccw.
            gsave clip 0 setgray 0 setlinewidth
            matrix setmatrix
            rotate dup scale                  %
            pathbbox newpath                  % x1 y1 x2 y2
            1 add cvi 4 1 roll                % y2' x1 y1 x2
            1 add cvi 4 1 roll                % x2' y2' x1 y1
            1 sub cvi exch 1 sub cvi    % x2' y2' y1' x1'
            2 copy exch translate
            3 1 roll sub                      % x2' x1' dy
            3 1 roll sub exch                 % dx dy
            { % repeat (dy) times             % dx
                        0 0 moveto dup 0      % dx dx 0
                        rlineto stroke        % dx
                        0 1 translate         % dx
            } repeat pop
            grestore
} bind def % L
```

Figure B.4.2 The PostScript Description of a Two-Input NAND Gate

```
/WW 1 def   % default Wire Width (space from line)
/Wto {                          % x1 y1 x2 y2 ==> x2 y2
%           Draws a path spaced WW from the line (x1,y1)–(x2,y2)
newpath
%           wire angle a=atan(dy/dx)
4 2 roll 4 copy             exch                        % x2 y2 x1 y1 x2 y2 y1
x1
3 1 roll sub 3 1 roll sub    % x2 y2 x1 y1 dy dx
atan dup 4 1 roll                         % x2 y2 a x1 y1 a
WW exch 90 add dup 180 add arc            % x2 y2 a
3 copy pop 3 2 roll                       % x2 y2 x2 y2 a
WW exch 90 sub dup 180 add arc            % x2 y2
closepath                                 % x2 y2  (leave for next Wto)
} bind def % Wto
/X {          % Draw an X on the current figure
              gsave clip
              pathbbox newpath            % x1 y1 x2 y2
              4 copy moveto lineto        % x1 y1 x2 y2
              3 1 roll exch               % x1 y2 x2 y1
              moveto lineto stroke        %
              grestore
} bind def % X% End of header code
%%EndProlog%%Page: 0:0
36 dup translate % margins
/Helvetica findfont 6 0.18 div scalefont setfont
0.18 dup scale % points/centi–micron
–800 2400 translate % cell_origin
.5 setgray
% CAA
400 700 1300 100 B fill
400 700 2900 100 B fill
2000 400 1300 –300 B fill
2000 400 1300 –1700 B fill
.3 setgray
% CPG
200 1800 1800 –1000 B fill
600 200 1800 –1200 B fill
200 1200 2200 –2400 B fill
200 3200 2600 –2400 B fill
0 setgray
```

Figure B.4.2—continued

```
% CCA
200 200 1400 -200 B fill
200 200 2200 -200 B fill
200 200 3000 -200 B fill
200 200 1800 -1600 B fill
200 200 3000 -1600 B fill
200 200 1400 500 B fill
200 200 3000 500 B fill
200 200 1400 -1600 B fill
0 setgray
% CWN
2600 200 1000 900 B 32 135 L
3000 1700 800 -800 B 32 135 L
% CMF
2000 400 1300 400 B 8 45 L
400 700 1300 -300 B 8 45 L
400 400 2100 -300 B 8 45 L
400 700 2900 -300 B 8 45 L
300 300 2200 -600 B 8 45 L
1100 300 2200 -900 B 8 45 L
300 400 3000 -1300 B 8 45 L
800 700 1300 -2000 B 8 45 L
400 400 2900 -1700 B 8 45 L
2000 400 1300 -2400 B 8 45 L
% CSN
800 600 1100 400 B 16 135 L
800 600 2700 400 B 16 135 L
1800 800 1700 -1900 B 16 135 L
% CSP
800 100 1100 300 B 16 45 L
800 100 2700 300 B 16 45 L
2400 800 1100 -500 B 16 45 L
600 800 1100 -1900 B 16 45 L
0 setgray
3100 -800 moveto (O) show
1900 -1100 moveto (A) show
2700 -1100 moveto (B) show
1900 -1900 moveto (GND!) show
2300 600 moveto (Vdd!) show
showpage
%%Trailer
%%Pages: 1
```

Figure B.4.2—continued

BIBLIOGRAPHY

[Adl79] Adleman, L., "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," in *Proceeding IEEE 20th Annual Symposium on Foundations of Computer Science*, pp. 55–60, 1979.

[Aga74a] Agarwal, R. C. and Burrus, C. S., "Fast One-Dimensional Digital Convolution by Multidimensional Techniques," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-22, No. 1, pp. 1–10, February, 1974.

[Aga74b] Agarwal, R. C. and Burrus, C. S., "Fast Convolution Using Fermat Number Transform With Applications to Digital Filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-22, No. 4, pp. 87–97, April, 1974.

[Aga75] Agarwal, R. C. and Burrus, C. S., "Number Theoretic Transforms to Implement Fast Digital Convolution," *Proceedings of the IEEE*, Vol. 63, No. 4, pp. 550–560, April, 1975.

[Aho74] Aho, A. V., Hopcroft, J. E., Ullman, J. D., *The Design and Analysis of Algorithms*, Reading, MA: Addison-Wesley Publishing Company, 1974.

[Ber68] Berlekamp, E. R., *Algebraic Coding Theory*, New York, NY: McGraw-Hill, 1968.

[Bla83] Blahut, R. E., *Theory and Practice of Error Control Codes*, Reading, MA: Addison-Wesley Publishing Company, 1983.

[Bla85] Blahut, R. E., *Fast Algorithms for Digital Signal Processing*, Reading, MA: Addison-Wesley Publishing Company, 1985.

[Blak84] Blake, I. F., Fuji-Hara, R., Mullin, R. C., and Vanstone, S. A., "Computing Logarithms in Finite Fields of Characteristic two," *SIAM Journal of Algebraic and Discrete Methods*, Vol. 5, No. 2, pp. 276–285, June 1984.

[Cmo87] *CMOS Cell Library Development Project*, Tampa, FL: University of South Florida, May, 1987.

[Con68] Conway, J. H., "A Tabulation of Some Information Concerning Finite Fields," in *Computers in Mathematical Research*, R. F. Churchhouse and J. C. Herz, Eds. Amsterdam: North-Holland, pp. 37–50, 1968.

[Coz85] Cozzens, J. H. and Finkelstein, L. A., "Computing the Discrete Fourier Transform Using Residue Number Systems in a Ring of Algebraic Integers," *IEEE Transactions on Information Theory*, Vol. IT-31, No. 5, pp. 580–588, September 1985.

[DEC90] *1990 DECWRL/Livermore Magic Release*, Palo Alto, CA: Western Laboratory. September, 1990.

[Ell82] Elliott, D. F., Rao, K. R., *Fast Transforms Algorithms, Analyses, Applications*, New York, NY: Academic Press, Inc., 1982.

[Gam85] Games, R. A., "Complex Approximations Using Algebraic Integers," *IEEE Transactions on Information Theory*, Vol. IT-31, No. 5, pp. 565–579, September 1985.

[Goo60] Good, I. J.,"The Iteration Algorithm and Practical Fourier Series," *J. Royal Statis. Soci.*, Ser. B20, pp. 361-372, 1958, Addendum 22, pp. 372–375, 1960.

[Har65] Hardy, G. H., Wright, E. M., *An Introduction to the Theory of Numbers*, Oxford, Claredon Press, 1965. (512 .81 H269i4)

[Hpd88] *HP DCS Manuals*, Colorado Springs, CO: Hewlett-Packard Company, 1988.

[Hub90] Huber, K., "Some Comments on Zech's Logarithms," *IEEE Transactions on Information Theory*, Vol. IT-36, No. 4, pp. 946–950, July 1990.

[Ima80] Imamura, K., "A Method for Computing Addition Tables in GF(p^n)," *IEEE Transactions on Information Theory*, Vol. IT-26, No.3, pp. 367–369, May 1980.

[Jen80] Jenkins, W. K., "Complex Residue Number Arithmetic for High-Speed Signal Processing," *Electronics Letters*, 14th, Vol. 16, No. 17, pp. 660–661, August 1980.

[Jen87] Jenkins, W. K., Krogmeier, J. V., "The Design of Dual-Mode Complex Signal Processors Based on Quadratic Modular Number Codes," *IEEE Transactions on Circuits and Systems,* Vol. CAS-34, pp. 354–364, April 1987.

[Kao85] Kao, R. S., "A Single Modulus Complex ALU for Digital Signal Processing," Master's Thesis, University of Florida, Gainesville, December 1985.

[Kao87] Kao, R. S., and Taylor, F. J., "Implementation of the Single Modulus Complex ALU," *Proceeding of the 8th Symposium on Computer Arithmetic*, pp. 21–27, May 1987.

[Kao90] Kao, R. S., and Taylor, F. J., "A Fast Galois Field Transform Algorithm Using Normal Bases," accepted for publication in *IEEE Proceeding of the 24th Asilomar Conference on Signals, Systems & Computers*, November 1990.

[Kao91a] Kao, R. S., "A Multiplier-Free Fast transform with Efficient VLSI Implementation for Polynomial RNS Processors," accepted for publication in *IEEE Proceeding of ICASSP*, May 1991.

[Kao91b] Kao, R. S., and Taylor, F. J., "The Basis-Change Algorithm for Fast Finite Field Transforms," accepted for publication in *IEEE Southeastcon '91*, April, 1991.

[Knu69] Knuth, D. E., *The Art of Computer Programming. Vol. II: Seminumerical Algorithms*, Reading, MA: Addison-Wesley Publishing Company, 1969.

[Kol77] Kolba, D. P., and Parks, T. W., "A Prime Factor FFT Algorithm Using High-Speed Convolution," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-25, No. 4, pp. 281–294, August 1977.

[Kro83] Krogmeier, J. V., Jenkins, W. K., "Error Detection and Correction in Quadratic Residue Number Systems," *Proceedings of 26th Midwest Symposium on Circuits and Systems*, pp. 408–411, August 1983.

[Kron79] Kronsjo, L. I., *Algorithms: Their Complexity and Efficiency*, New York: John Wiley & Sons, 1979.

[Lid86] Lidl, R. and Niederreiter, H., *Introduction to Finite Fields and Their Applications*, Cambridge, MA: Cambridge University Press, 1986.

[Lin83] Lin, S., Costello, D. J. Jr., *Error Control Coding Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[Lip81] Lipson, J. D., *Elements of Algebra and Algebraic Computing*, Reading, MA: Addison-Wesley Publishing Company, 1981.

[Leu81] Leung, S., "Application of Residue number systems to Complex digital Filters," *IEEE Proceeding of the 15th Asilomar Conference on Signals, Systems & Computers*, November 1981.

[Mac77] MacWilliams, F. J. and Sloane, N. J. A., *The Theory of Error-Correcting Codes*, Amsterdam: North-Holland, 1977.

[McC79] McClellan, J. H. and Rader, C. M., *Number Theory in Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1979.

[McE87] McEliece, R. J., *Finite Fields for Computer Scientists and Engineers*, Norwell, MA: Kluwer Academic Publishers, 1987.

[Mea80] Mead, C. and Conway, L., *Introduction to VLSI Systems*, Reading, MA: Addison-Wesley Publishing Company, 1980.

[Mul89] Mullin, R. C., Onyszchuk, I. M., Vanstone, S. A., and Wilson. R. M., "Optimal Normal Basis in GF(p^n)," *Discrete Applied Mathematics*, Vol. 22, pp. 149–161. 1989.

[Nag64] Nagell, T., *Introduction to Number Theory*, New York: Chelsea, 1964.

[Niv80] Niven, I., Zuckerman, H. S., *An Introduction to the Theory of Numbers*, New York: John Wiley & Sons, 1980.

[Nus82] Nussbaumer, H. J., *Fast Fourier Transform and Convolution Algorithms*, 2nd ed. Berlin: Springer-Verlag, 1982.

[Opp75] Oppenheim, A. V., and Schafer, R. W., *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.

[Pet72] Peterson, W. W., Weldon, E. J. Jr., *Error Correcting Codes*, Cambridge, MA: MIT Press, 1972.

[Pol50] Pollard, H., *Theory of Algebraic Numbers*, Carus Math. Monographs, No. 9, Math. Ass. Amer., 1950.

[Rad68] Rader, C. M., "Discrete Fourier Transforms When the Number of Data Samples Is Prime," *Proceeding IEEE*, Vol. 56, pp. 1107–1108, June 1968.

[Rad72] Rader, C. M., "Discrete Convolution via Mersenne Transforms," *IEEE Transactions on Computers*, Vol. C-21, No. 12, pp. 1269–1273, December 1972.

[Rao81] Rao, T. R. N., "Arithmetic of Finite Fields," *Proceeding of the 5th Symposium on Computer Arithmetic*, pp. 2–5, May 1981.

[Red86] Redinbo, G. R. and Rao, K. K., "Expediting Factor-Type Fast Finite Field Transform Algorithms," *IEEE Transactions on Information Theory*, Vol. IT-32, No. 2, pp. 168–194, March 1986.

[Ree75a] Reed, I. S. and Truong, T. K., "The Use of Finite Fields to Compute Convolutions," *IEEE Transactions on Information Theory*, Vol. IT-21, No. 2, pp. 208–213, March 1975.

[Ree75b] Reed, I. S. and Truong, T. K., "Complex Integer Convolutions Over a Direct Sum of Galois Fields," *IEEE Transactions on Information Theory*, Vol. IT-21, No. 6, pp. 208–213, November 1975.

[Sch86] Schroeder, M. R., *Number Theory in Science and Communication*, Heidelberg: Springer-Verlag, 1986.

[Ska87] Skavantzos, A., "The Polynomial Residue Number System and Its Applications," Ph.D. dissertation, University of Florida, Gainesville, 1987.

[Sod86] Soderstrand, M. A., Jenkins, W. K., Jullien, G. A., and Taylor, F. J., eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* New York, NY: IEEE Press, 1986.

[Tay81] Taylor, F. J., "Large Moduli Multipliers for Signal Processing," *IEEE Transactions on Circuits and Systems*, Vol. CAS-28, No. 7, pp. 731–736, July 1981.

[Tay82] Taylor, F. J., "A VLSI Residue Arithmetic Multiplier," *IEEE Transactions on Computers*, Vol. C-31, No. 6, pp. 540–546, June 1982.

[Tay83] Taylor, F. J., "An Overflow-Free Residue Multiplier," *IEEE Transactions on Computers*, Vol. C-32, No. 5, pp. 501–504, May 1983.

[Tay84] Taylor, F. J., "Residue Arithmetic: A Tutorial with Examples," *IEEE Journal Computers*, Vol. 17, No. 5, pp. 50–63, May 1984.

[Tay85] Taylor, F. J., Papadourakis, G., Skavantzos, A., and Stouraitis, A., "A Radix-4 FFT Using Complex RNS Arithmetic," *IEEE Transactions on Computers*, Vol. C-34, No. 6, pp. 573–576, June 1985.

[Van78] Vanwormhoudt, M. C., "Structural Properties of Complex residue Rings applied to Number Theoretic Fourier Transforms," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. ASSP-26, No. 1, pp. 99–104, February 1978.

[Veg76] Vegh, E. and Leibowitz L. M., "Fast Complex Convolution in Finite Rings," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. ASSP-, No. 4, pp. 343–344, August 1976.

[Wan85] Wang, C. C., Truong, T. K., Shao, H. M., Deutsch, L. J., Omura, J. K., and Reed, I. S., "VLSI Architectures for Computing Multiplications and Inverses in GF(2^m)," *IEEE Transactions on Computers,* Vol. 34, No. 8, pp. 709–717, August 1985.

[Wes85] Weste, N., and Eshraghian, K., *Principles of CMOS VLSI Design A System Perspective*, Reading, MA: Addison-Wesley Publishing Company, 1985.

BIOGRAPHICAL SKETCH

Rom-Shen Kao was born in Penhu, Taiwan, on October 19, 1955. He received a BSEE at the National Chiao-Tung University, Hsin-Chu, Taiwan, in 1979. From October 1979 to August 1981, he served in the Chinese Air Force as an electronics officer. In 1981, he joined the Electronics Research and Service Organization, Hsin-Chu, Taiwan, as a microcomputer hardware design engineer. He entered the graduate school at the University of Florida in August 1983 and received a MSEE in 1985. From 1985 to 1987, he worked as a design engineer at the Vital Industrial, Inc., Gainesville, Florida, where he was involved in the development of video post production equipments. In 1987, he joined the Digital Services Corporation, Gainesville, Florida, as a video signal processing engineer. He is currently completing his Ph.D in electrical engineering at the University of Florida. He is scheduled to receive his Doctor of Philosophy degree in May of 1991.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Fred J. Taylor, Chairman
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Y. C. Chow
Professor of Computer and Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Herman Lam
Associate Professor
of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Mark E Law

Mark Law
Assistant Professor
of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Jose Principe
Associate Professor
of Electrical Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

May 1991

for Hubert A. Bevis

Winfred M. Phillips
Dean, College of Engineering

Madelyn M. Lockhart
Dean, Graduate School